Australia A $ 4.75 | New Zealand NZ $ 6.50
Singapore S $ 9.45 | Hong Kong H $23.50
Malaysia M $ 9.45 | Sweden 30:-SEK

$2.95 USA

68' MICRO JOURNAL

GIMIX-Macintosh-SWTPC-CoCo

VOLUME VII ISSUE VI • Devoted to the 68XX User • June 1985

"Making Small Computers Do Big Things"

0 74470 12810 06

WE DON'T PLAY GAMES

X-12+ A SERIOUS COMPUTER IN A DESKTOP PACKAGE

Multiprocessor Technology - Combination of 8, 16 and 32 bit types

1.0 Megabyte Memory - Insures no limitation on programs

"Winchester" Disk System - Fast response, large storage capacity

UniFlex* Operating System - The standard of comparison

Hardware Floating Point - Unmatched speed in a small system

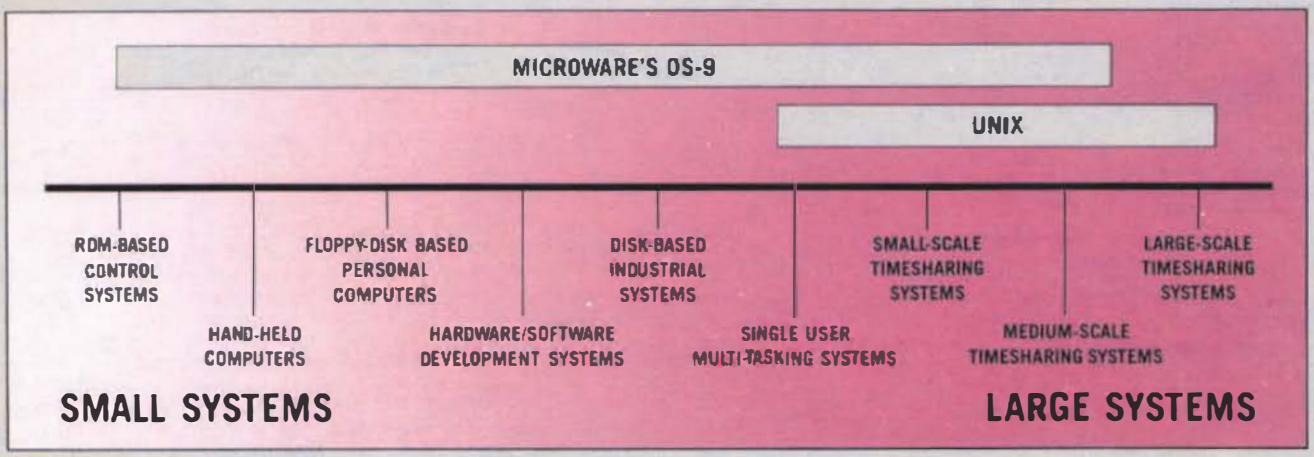
Up to Three Terminals - Instant expansion

*Trademark of Technical Systems Consultants

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum



Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delivers outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit systems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser systems.

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technology is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port system or application software back and forth to standard Unix systems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a pleasure.

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating systems. It multiplies programmer productivity and memory efficiency. Application software can be built from individually testable software modules including standard "library" modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported by many independent software suppliers.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

microware® OS-9™

MICROWARE SYSTEMS CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
Phone 515-224-1929
Telex 910-520-2535

Microware Japan, Ltd
3-8-9 Baraki, Ichikawa City
Chiba 272-01, Japan
Phone 0473(28)4493
Telex 299-3122

'68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS - HARDWARE
Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S09 - 5/8 DMF Disk - CDS1 - 8212W - Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS - WORD PROCESSORS
Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

Great Plains Computer Co., Inc.
PO Box 916
Idaho Falls, ID 83401
Stylograph - Mail Merge - Spell

CONTENTS

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, Tn. 37343

Phone (615) 842-4600 or Telex 558 414 PVT BTH

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, Tn. and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, Tn. 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add $9.50 a Year. Other Foreign Add $12 a Year for Surface, Airmail Add $48 a Year. Must be in U.S. currency.

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be on either 5 or 8 inch disk in STYLOGRAPH or TSC Editor format with 3.5 inch Column width. All disks will be returned. Articles submitted on paper should be 4.5 inches in width (including Source Listings) for proper reductions. Please Use A Dark Ribbon!! No Blue Ink!! Single space on 8X11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands ONLY should be used: .sp space, .pp paragraph, .fi fill and .nf no fill. Also please do not format within the text with multiple spaces. We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except .pg page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and must be signed. Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

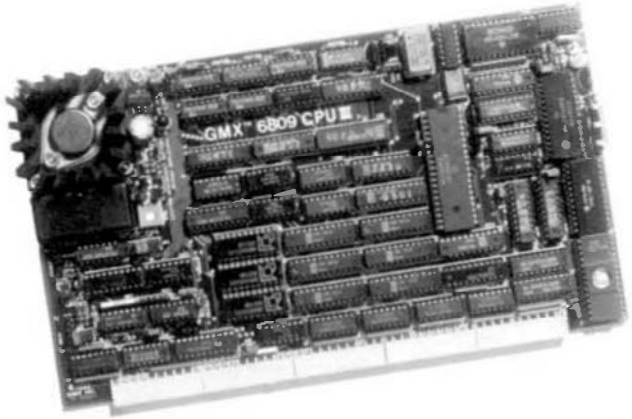
Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of $9.50 for first 20 words and .45 per word after 20. All classifieds must be paid in advance. No classified ads accepted over the phone.

GIMIX STATE OF THE ART 6809 SYSTEMS FOR THE SERIOUS USER.



GIMIX has 19MB or high performance 47MB Winchester Drive Systems and/or Floppy Disk Drive Systems.

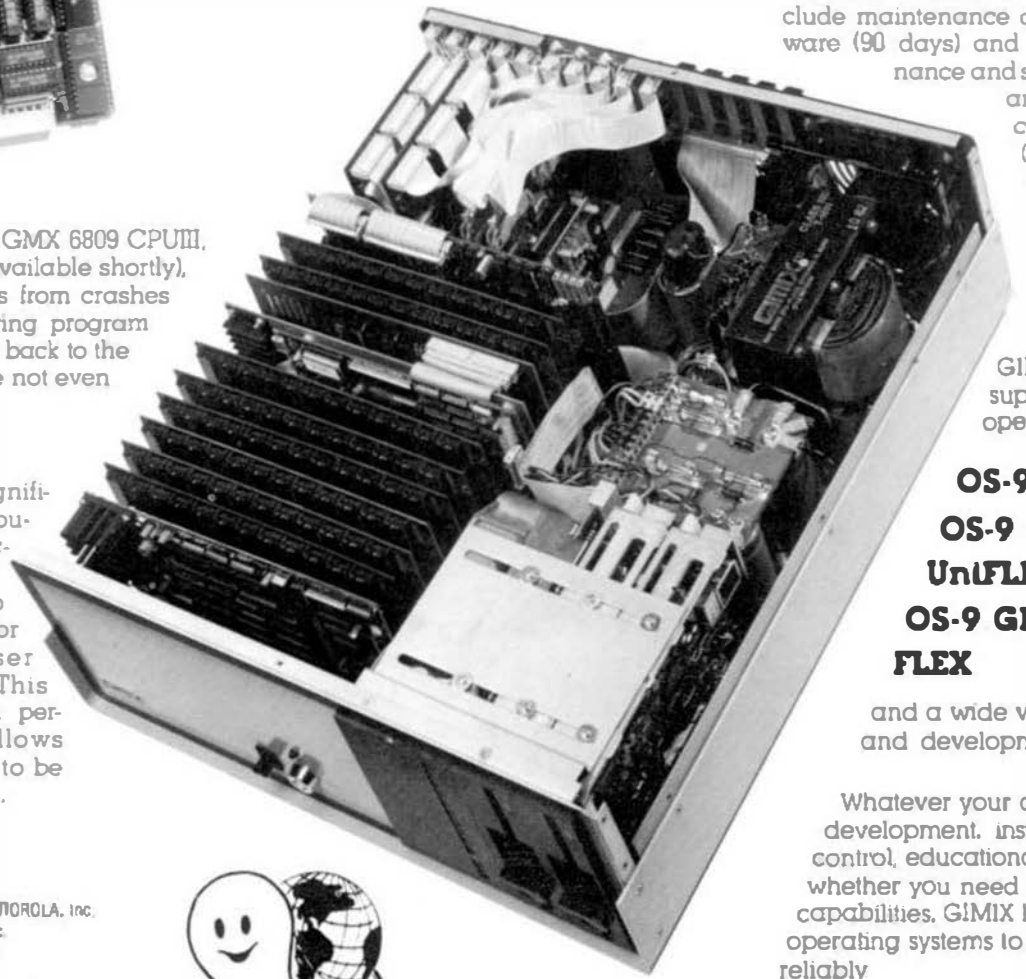
For the user who appreciates the need for a bus structured system using STATIC RAM and powered by a ferro resonant constant voltage transformer.

GIMIX has single user systems that can run both FLEX and OS-9 or Multi user systems for use with UniFLEX or OS-9.

GIMIX versions of OS9 and UniFLEX include maintenance and support by Microware (90 days) and TSC (1 year). Maintenance and support after this period are available at extra cost.

(NOTE: this support and maintenance is only for use with approved GIMIX hardware)

For the ultimate in performance, the Unique GMX 6809 CPUIII, using either OS-9-GMXIII or UniFLEX GMXIII (available shortly), gives protection to the system and other users from crashes caused by defective user programs. e.g. During program development, a programmer who crashes goes back to the shell or the debugger, while the other users are not even aware anything occurred.

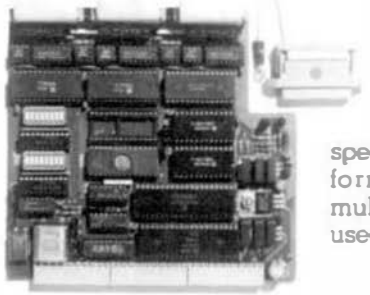


GIMIX 6809 systems support five predominant operating systems:

OS-9 GMX III,
OS-9 GMX II,
UniFLEX,
OS-9 GMX I,
FLEX

and a wide variety of languages and development software.

The intelligent serial I/O processor boards significantly reduce system overhead by handling routine I/O functions, thereby freeing up the host CPU for running user programs. This speeds up system performance and allows multiple terminals to be used at 19.2K baud.



Whatever your application: software development, instrumentation, process control, educational, scientific or business; whether you need single or multi-user capabilities, GIMIX has hardware and the operating systems to get the job done reliably.

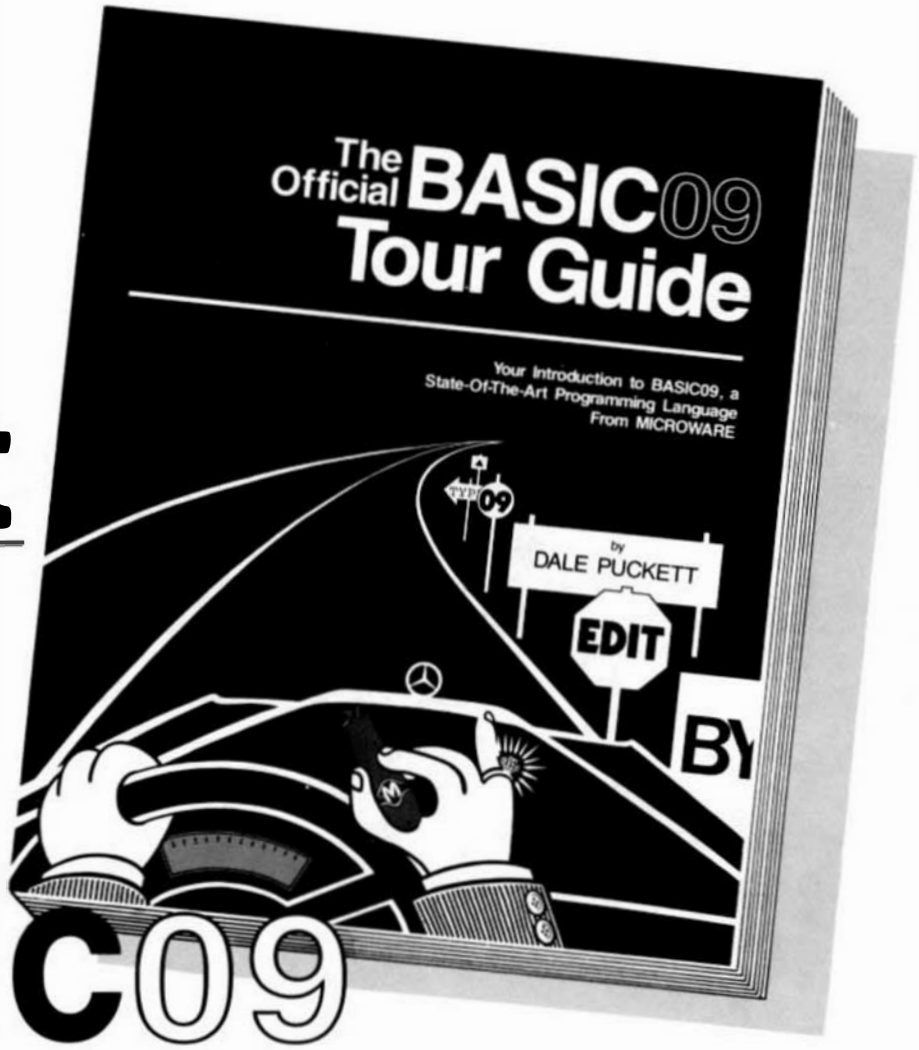
Please phone or write if you need further information.

BASIC-09 and OS-9 are trademarks of Microware Systems Corp. and MOTOROLA, Inc.
FLEX and UniFLEX are trademarks of Technical Systems Consultants, Inc.
GIMIX, GHOST, GMX, CLASSY CHASSIS, are trademarks of GIMIX, Inc.

GIMIX inc.

1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

Get the most out of BASIC09



The **OFFICIAL BASIC09 TOUR GUIDE** is skillfully written in a friendly and easy-to-read style. Just perfect for those new to computers and to BASIC09. It's also a *valuable reference book* for programmers, engineers, students and hobbyists, providing an in-depth look at BASIC09 plus an overview of the OS-9 operating system. Comprehensive reference sections on BASIC09 and OS-9 commands are also included.

The book "maps" out your route through the Mercedes of Basics . . . BASIC09 and puts you in the driver's seat in no time. Fasten your seatbelt, sit back and enjoy the ride to perfecting your programming skills.

MICROWARE . . .

The **OFFICIAL BASIC09 TOUR GUIDE** comes from the people who wrote BASIC09. As the leader in 6809 system software, we at MICROWARE care about our users and want to help you get the most from our products.

It's Easy to Order.

Phone orders are accepted from MasterCard or VISA cardholders or for COD shipment. You can also order by mail using the coupon below. Quantity discounts are available to educational organizations and dealers. For further information contact Microware.

Microware Systems Corporation
1866 N.W. 114th Street
Des Moines, Iowa 50322
Telephone 515/224-1929
Telex 910-520-2535

Please send _____ copies of the **Basic09 Tour Guide** book at $18.95 each. Add $2.00 for UPS shipping in the U.S. or $5.00 for overseas air mail per book. Iowa residents add 4% sales tax.

Name _____

Address _____

City _____

State _____ Zip _____

☐ I have enclosed a check

☐ Charge to my bank card:
MasterCard ☐ VISA ☐
Card Number _____
Expiration _____

microware®

Specialists in system software for 68-family microprocessors since 1977.

OS-9 and BASIC09 are trademarks of Microware and Motorola

FLEX™ USER NOTES
THE 6800-6809 BOOK

By: Ronald W. Anderson
As published in 68 MICRO JOURNAL™

The publishers of 68 MICRO JOURNAL are proud to announce the publication of Ron Anderson's **FLEX USER NOTES,** in book form. This popular monthly column has been a regular feature in 68 MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. Now all his columns are being published, in whole, as the most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

As a SPECIAL BONUS all the source listing in the book will be available on disk for the low price of: FLEX™ format only — 5" $12.95 — 8" $16.95 plus $2.50 shipping and handling, if ordered with the book. If ordered separately the price of the disks will be: 5" $17.95 — 8" $19.95 plus $2.50 shipping and handling.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC
	(And many more)

**Over 30 TEXT files included in ASM (assembler) — PASCAL — PIC (position independent code) TSC BASIC-C, etc.

NOTE: .C1,.C2, etc. = Chapter 1, Chapter 2, etc.

This will be a limited run and we cannot guarantee that supplies will last long. Order now for early delivery.

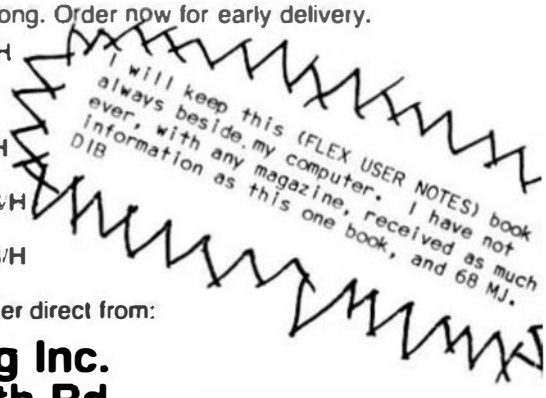
Foreign Orders Add $4.50 S/H

Softcover — Large Format

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H



See your local S50 dealer/bookstore or order direct from:

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4601

TELEX 558 414 PVT BTH

'™FLEX is a trademark of Technical Systems Consultants

FLEX User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, Mi 48105

Fold Them Up Again

This week I have received two 5" disks in the mail. The first was from a reader who wanted to send me some of his efforts as a little appeasement for a letter he had written to '68' Micro Journal about this column. (More about this below). The second disk was from a customer who wants me to rewrite some math routines for him. They had one thing in common. Both were neatly folded in half by the mailman or the U.S. Postal service. Each was mailed with a letter in a 9 by 12 envelope, and each was "protected" by a piece of cardboard (scarcely stiffer than the envelope itself) on each side. Neither disk is readable as received.

I have a neat solution for this problem, since I have run across it before. The solution is simply to open the disk jacket very carefully and remove the inner magnetic recording disk. I have an empty non-folded jacket from an old disk whose magnetic surface had been destroyed, and that jacket has been carefully slit along the back edge. Now I place the "innards" of the damaged disk into the open but otherwise undamaged jacket, and voila, I can read the disk. When I do this, I always copy the contents of the damaged disk to a good one, immediately, and then throw away the damaged disk.

Now, perhaps this is the time to describe how to mail a 5 inch disk so it arrives in usable and useful condition. First of all, the cardboard from the back of a scratchpad, or "shirt cardboard" is NOT suitable. Go find a sturdy corrugated carton and cut two squares from it that are just larger than the disk jacket. Now turn one of them at right angles to the other, so that the corrugations in one piece run crosswise to the corrugations in the other. Corrugated board is easy to crease in the direction of the corrugations and much stiffer crosswise of them. Crossing the corrugations makes the combination strong in both directions.

Place your disk between them and tape them together securely. If you cut the squares just large enough, the disk with its protection will fit snugly in a 6 by 9 envelope (available at any office supply store and a number of other national stores such as K-Mart.) The 6 by 9 envelope will also hold a letter folded in half. The advantage of using a smaller envelope (rather than a 9 by 12) is that it will fit most mailboxes without tempting the mailman to fold it in half. If you want to spend more on a mailer than the disk cost, you can buy special mailers, but they won't protect the disk any more than the above technique. Corrugated board may be cut with an "Exacto" knife or a "Stanley" Utility knife, or on most large paper cutters. I seriously doubt that it will do any good, but you might want to write "Magnetic Media... Do Not Bend" on the envelope. I suspect some postal workers take that as a challenge to see if they can wreck the contents.

Operating Systems

A few years ago I noticed an ad for a program to run under CP/M that would act as a "buffer" or "interface" between the user and CP/M. In other words, it would be more "user friendly" and would translate commands that were more understandable to the user, to those understood by CP/M. At the time I wondered why CP/M had become the most popular operating system. We FLEX users certainly don't need the help of a program to translate our simple commands for it.

Well, the other day, what I would call a product announcement arrived in my mail. I don't want to pick on anyone, so I won't mention the company of origin, but it is essentially the program mentioned above, but written to make OS-9 more understandable to the user. I have

nothing but good wishes for the company that wrote the software. I hope they do very well. However..... In my not so humble opinion (Let's face it, I have strong opinions on some things), if you need the complexities of a multi user and multi tasking operating system, you ought to take the time to learn enough about it to be able to use it effectively without crutches. If you don't need such a system, you ought not to have it, but enjoy simplicity with a single task single user system like FLEX! Or Star-DOS.

Complaint Department

Now and then I get a letter of complaint. I mentioned a disk arriving folded in half at the start of this column. The writer indicated that he had written a letter to '68' about my column, and he enclosed a copy for me. Essentially he said that he was sick and tired of my writing about this compiler and that compiler, and why this one was better than that. Why don't I spend some time writing about FLEX per the title of the column. At this point I had prepared some answers to that question, but a call to Don Williams brought me the information that a number of readers have been asking for some basic information on FLEX. I'm willing to devote a portion of this column each month to answering some specific questions about FLEX. I've been using it so long that I can't imagine what questions would be asked. This time I'll make a guess and talk about the various character output routines in FLEX a little farther along. For now, I'll continue with some observations I was going to make about my column effort in general.

This column has really been a struggle for me at this point. I have something to say, but I can't figure out just how to say it. Please bear with me in what follows. I've tried three or four times to rewrite it and I still don't like it.

At the risk of getting several hundred letters, here goes. First of all, perhaps I ought to change the name of the column. I'm open to suggestions. Maybe I've been running the SS-50 system for too long now, but frankly if I see another CAT or COPY utility for FLEX published in '68', I'm going to do one of three things.

1. Burn that issue while sticking pins in a Don Williams Doll;
2. Get sick all over it;
3. Cancel my subscription to '68' Micro Journal.

Sure, I have Leo and Bruno's CAT and COPY utilities, but to be perfectly honest, I have no trouble with the original FLEX supplied versions of either. I found those adequate for a long time, at least until the TSC utility package with DIR came along. It would seem that some FLEX "users" never want to get beyond enhancing the operating system. As I said to one admitted "hacker" (his term, not mine), if you find great pleasure in fine tuning an operating system as a hobby, fine. Go to it and enjoy it.

I think of myself as a FLEX USER not a FLEX TINKERER. A computer, I've said before, is a tool, not an end. I'll make the same statement about an operating system. What I like best about FLEX is that it is so unobtrusive. I can easily write code that runs on my system independent of FLEX. Then I can stuff that code in a ROM and run a little two or three board system in a dedicated application.

Obviously my uses of my computer are not the same as everyone else's. I've made that statement so often lately that I think I'm beginning to sound like a broken record. The problem is that I have to write from where I am, if you will pardon poor grammar.

The point might well be made that many of the readers of '68' are new to computing and as such are not familiar with FLEX. Frankly I'd have a very hard time writing a FLEX beginner's column. However, I am quite willing to answer questions regarding FLEX that I think would be of general interest to new users as part of my column each month, at least for a while.

One question the writer of the letter brought up is that of how the DAT works. I could go into that some time for a column, (realize that this is essentially a hardware question) but I think it is a moot point. The only uses for the DAT with FLEX presently are for "Virtual Disk" and one utility that I have seen that allows installation of FLEX utilities in RAM, called LOCAL. At this point, with the big switch to IBM, its me-too copies, and the Mac, no supplier in his right mind is going to rewrite any software written to run under FLEX to take advantage of extended memory (though it would certainly be nice for a text editor to have an extra large edit buffer).

Maybe I'm feeling a little like old Charlie today. Charlie and Ruth ran a little short order restaurant coffee shop across the street from the main part of the campus of the University of Illinois when I went there to work about 20 years ago. They were holding out for the U to expand and buy their property, of course, (which it did eventually) but the place was pleasant, and well populated by university staff for coffee breaks. They were, however, a little independent. One day someone complained about the french fries. Charlie was heard to reply, "If you don't like them, don't eat them!". At that point, he turned away and continued about his business.

I guess that about expresses my feelings about the column. I'll add a qualification to that, however. I'll say if you don't like it but won't communicate to me what you would like me to discuss in it, don't expect it to change very much. I've asked for reader response before, and never received more than three letters when I've done so, generally all three with different ideas about what should be in the column. Without input from you readers, I'm going to have to write about where I am right now.

I tired long ago of writing FLEX utilities and went on to more interesting computer pursuits (at least more interesting to me). I don't need fancy utilities that take the place of the simple ones I have. I don't need tree structured directories or "submit" utilities that let me create a file of commands for the computer. I need only a good editor, an assembler, and a good compiler that allows me to write complex programs that work, in the shortest possible time. I continue to evaluate new software in the way of compilers, editors, assemblers, as they become available, just in case something comes along that is significantly better than what I am now using. Maybe you could describe me, because of that, as a "language hacker". Rather than dwell on operating system and utility improvements, I dwell on the possibility that a better language or more efficient compiler will come along. (Meanwhile, I use what I have now to its fullest capabilities).

As I get involved with new equipment as I did recently with an IBM clone, I am interested to see if any of this new equipment works any better or faster than what I have now. Therefore I test and report the results of the testing.

Part of my problem is that I don't like the trend in computer publications away from "how to program" toward "how to use XYZstar". A walk through the "Computer" section of the bookstores shows me that the books on programming in PASCAL or "C", and subjects like Compiler Design, have been relegated to one or two shelves, while books on "Understanding Wordstar" or "Getting the most out of Visicalc" have taken over most of the space. The popularization of personal computers has shifted the emphasis of the available books from how to WRITE a program in general, to how to RUN one program in particular. I think that is too bad. I'll go further than that. I'd like to do my part to buck that trend. If you believe the ads in Byte, an XYZ computer that comes complete with a generic spreadsheet program and a generic word processor is a complete system. Come on now, if you believe that a personal computer can only do word processing and run a spreadsheet, you're missing something. Most of the fun of having a computer is programming it to do what you want it to do, not running someone else's software.

My advice in general would have to be that, if you want to learn all about FLEX, read the programmer's manual. If you don't understand that, buy a copy of "6809 Assembly Language Programming" by Lance Leventhal, published by Osborne and Associates. By the time you have read and understood that book, the programmer's manual will make sense without any outside help. If you don't understand Assembly Language programming, not much of any explaining I could do will make any more sense to you than the programmer's manual does now.

Now having said my piece, I'll make a plea for some guidance from the readers of this magazine. If you have a suggestion for a topic to be discussed here, send me a letter. I presently have some suggestiona from two of you, that I should discuss some topics related to the insides of FLEX itself. If I don't get more than a dozen letters, I'll assume that most of you are perfectly happy with what I am doing now, and I'll continue to do it just this way. If I do get more than a dozen letters, and any two or three agree on a topic, I just might be pursued to do something with it. If the response is interesting, I'll report the results here in the future. As I've said above, past requests for reader input have never

resulted in more than three letters of response.

If my "independent attitude" has made you angry, GOOD. Perhaps that will stir more than two or three of you to respond! You can write me directly at the address at the top of this column. Letters to '68' Micro Journal will be delayed since they have to be relayed to me here in Ann Arbor.

STAR-DOS Update

Peter Stark tells me that he is busily testing his upgrade of STAR-DOS that handles random files. He has recently tailored a version of STAR-DOS to use the clock chip on the PT69 system for the date and time information. All you need to do is supply three AA cells for battery backup of the clock chip and STAR-DOS will come up with the date registers properly loaded from the clock chip. It will also read the time and insert it in the directory record of any file you write while running STAR-DOS. Of course that includes any files created or edited, and output files from compilers, assemblers, etc. I've mentioned the TCAT utility that lists the files on a disk in order from newest to oldest.. Handy for finding the file you were working on this morning at 2:00 and the name of which you can't remember.

I also recently received an improved STAR-DOS from Peter with a couple recently discovered bugs eradicated. Someone else and I almost simultaneously reported a problem using a printer from an Extended BASIC program, and Peter has removed the problem.

Editors

I had a discussion with one of the software suppliers some time ago regarding the control or function keys for a screen editor. I maintain that anyone who is a touch typist (i.e. either learned how to type before becoming a computer terminal user or learned to type correctly afterwards, using all the fingers of both hands, and types without looking at the keyboard) would vastly prefer to do cursor control and other functions by way of the control key. I sometimes realize that I have hit the wrong key, type control H, and then the correct key without looking at either the CRT or the keyboard. The backspace key on the keyboard I am using to write this, is four keys away (diagonally) from the little finger of my right hand. If I go for it, I won't return to my "home keys" in the middle row without looking at the keyboard. The control key is two keys left from the "A" key (I'd prefer it to be next to the A, as in the old ADM-3A), and I can reach it with my little finger without looking.

The supplier in question is using a terminal with ten or twelve function keys a row above the number keys. Fooey! Give me a nice pattern of keys for cursor motion that I can reach without looking, and I can zip along while paying strict attention to the text that I am typing from my rough draft notes. Make me look at function keys and I am slowed down by a factor of 2 or 3.

I note a recent Radio Shack ad for Wordstar with some interest. The ad says that touch typists prefer control codes to function keys for their text editing. Wordstar lets you have it both ways (or either way). Hooray for Wordstar and all those who understand how a typist operates, and Boo-Hiss to the others that think function keys are so great. I'm not saying that function keys are useless in something like a CAD system or perhaps a general ledger package in which each one will bring up a different menu or function, just that someone typing text ought to be able to control the cursor without looking at the keyboard. Even here, though, some thought ought to be given to an alternate code. The Tandy 1200 Microsoft BASIC has a menu across the bottom line (very distracting since program is entered on the line just above it). You can find such interesting things as a list of the function keys and what they do. For example F3 types 'load"' on the screen. Before I can remember that it was F3, let alone find and operate the F3 key, I can type 'load"'and the name of the file to be loaded!

If you spend a significant fraction of your time in front of a terminal and you don't touch type, you owe it to yourself to get one of the available typing tutor programs and spend a few hours a week with it. If 30 hours of practice improve your typing speed so that you can save 5 hours a week, you'll break even in 6 weeks and be ahead for the rest of your computer career or hobby. Besides, once you learn to touch type, you'll side with me against the use of function keys for editing!

Outch and Outch2

Now having said my piece, by popular demand let's look at FLEX just briefly for

this time. Next time perhaps I will have a little more on the subject.

Flex takes over some of the interface job from the programmer. For example you don't have to write software to manipulate the 6850 serial interface chip directly in order to output characters to your terminal. Flex itself makes use of output routines in the Monitor ROM. It builds upon these so that it can include nice features such as the PAUSE. When a program is dumping a lot of data to the screen, you can configure FLEX via the TTYSET utility to stop every 22 or 24 lines and wait for you to hit ESCAPE to allow data flow again. This feature is built in to the output routine in FLEX called PUTCHR. You simply load the A accumulator with the Ascii value of the character you want to output, and do a JSR PUTCHR (which happens to be at address $CD18).

Maybe you've noticed a particular feature of FLEX while you have been assembling or compiling a program. You have selected the option of sending the listing to the printer and yet an error message appears on your terminal. FLEX has a couple of possible output routine entry points. When FLEX is initialized by a "Warmstart" (Jump to address $CD03) using either of these two routines called OUTCH and OUTCH2 causes the output of a character to the terminal. If you've transfered output to the printer, a jump to OUTCH will output the character to the printer, but a jump to OUTCH2 will always output to the terminal. While we are at it, lets discuss some terminology that the UNIX folks like to throw around. Your terminal is called the "Standard Output". Switching output to your printer or a disk file is called "Output Redirection". We would therefore say that OUTCH is capable of output redirection while OUTCH2 always outputs to the standard output.

Now it is not quite that simple. You can access these two routines directly, or you can access FLEX at a higher level. The main "output a character" routine in FLEX that we discussed above, PUTCHR. As I mentioned above, you load accumulator A with the Ascii code of the character you want to output and do a JSR PUTCHR. Location $CC22 in FLEX is called "Output Switch". If $CC22 contains $00, the output of PUTCHR is through OUTCH so it will go wherever the output is currently directed, i.e. Printer, terminal, or file. If $CC22 contains a non zero value (any code other than $00) the PUTCHR routine will use OUTCH2. A program (Utility or otherwise) might make use of this by clearing or complementing OUTSW (My name for $CC22) to direct the program output unconditionally to the terminal, for example for output of error messages. FLEX initializes OUTSW to 00, so PUTCHR outputs through OUTCH unless you "fiddle" with it. I've used the technique in some utilities that I wrote some time ago, but I have not seen a great deal of use of this nice feature by many of the software suppliers. FLEX has a higher level routine called PSTRNG, at $CD1E. You point the X index register at the start of a string of text in memory and to a JSR PSTRNG. PSTRNG outputs characters until it finds a $04 value
and then returns. This routine will honor OUTSW, so you can direct a message to the terminal even though output has been redirected to the printer.

```
Message  FCC /THIS IS A MESSAGE./
         FCB $04
DOMESG   LDX #MESSAGE
         JSR PSTRNG
         RTS
```

You Position Independent Code purists would use LEAX MESSAGE,PCR in the above code to maintain position independence.

FLEX has an Input Switch also, which does essentially the same thing allowing input from a file or the terminal under program control. These routines and what they do, are documented in the FLEX Advanced Programmer's Guide that is supplied with each copy of FLEX.

I hope this discussion convinces some of you that you have to have some understanding of Assembler programming before you are going to know a great deal about how FLEX works.

Would any of you like some discussions of how to structure a program reasonably in assembler, and proper use of subroutines? I see lots of code that involves what I consider to be incorrect use of subroutines. Many times subroutines are very general when they need to be specific, and at other times there are too many specific ones where one general one would do very nicely.

This has been an exceptionally hard column for me to write. I've worked the center section about the complaint over and over, and it would seem to me that it gets more confusing rather than better, so I am going to stop. I really would like some reader input. Please write!

OS-9 User Notes

Peter Dibble
517 Goler House
Rochester, N.Y. 14620

Doesn't Anyone Else Work Late

I've watched the sun rise while I programmed several times in the last few months. The first time that happened while I was working with OS-9 I discovered something. The "make" program I published here several months ago, had a bug in it. The defect only shows up between midnight and 1:00 AM. Between those hours the hour byte in the record returned by gettime is zero. I used strncpy to copy the date/time information. Strncpy sees the zero byte as the end of a string and stops copying there. It fills the rest of its output up to the specified length with nulls. Because of this, make thought all times from midnight until 1:00 AM were midnight. The results were disconcerting.

The solution is to use the _strass function to copy date/times. Unlike the string copy functions, _strass copies a specified number of bytes without looking at them. If you don't use Microware C I'm afraid you may need to write a special function; _strass isn't part of standard C.

I assume someone else must have tried to use make in the early morning. You should have sent me a note. I don't mind hearing that my programs aren't always perfect. I'm embarrassed that it took me this long to find and fix the problem.

I also overlooked something in the series of directory listing programs I published. In the directory, each file name is terminated by a byte with the high-order bit set. I printed them just as they stood in the directory. Under the right circumstances those high-order bits look funny. They haven't caused me any trouble beyond making the last character in each file name appear in my terminal's secondary font so I haven't fixed done anything about it.

If you get strange results from my directory programs, try masking the high-order bit out of the last byte in each file name.

OS-9 68K -- A Look at the Future

I've just had a chance to read the OS-9 68K manuals. I've been dreaming about getting a 68K system, but the manuals aren't interesting only in that light.

It is Microware's stated policy to keep OS-9 and OS-9 68K as compatible as possible. This specifically means that they will update OS-9 with as many of the improvements they invented for OS-9 68K as possible. There are a BUNCH of interesting changes that we might see.

The 68000 supports a much larger address space than the 6809. This let Microware write utility programs in C and add features. It's easy to move C programs between the two environments for OS-9, but memory constraints may rule out some of the nicer features. For example, I would be surprised if we got wild card matching for our shell.

Two other enhancements to the shell seem more portable: error file support and command priority specification. Both these features could be done simply in assembler for the 6809.

Error Messages

Very early Level Two users got an error message file in their SYS directory, but no command to bring the file into use. The problem was with the multi-user Level Two environment. The error message/number switch was a feature of a system call.

There was no easy way to get OS-9 to treat F$PErr calls coming from one user differently from the same call from another user. Microware thought that each user should be able to choose whether he wanted to receive error messages or just numbers, so the problem sort of sat there.

They made a big change for OS-9 68K. The F$PErr call takes a error message path as an argument. Each time a program issues an error message it can choose an error file (or no error file). The most obvious result of this change is that the choice between error messages and error numbers is made with a shell option, not with a command. Since each user has his own shell, each user can make his own choice about error messages. It also means that programs can come with their own error message files and still use the standard error message interface.

Setting Process Priority

The priority-specification command line option is a useful feature, but is less far-reaching than the error message thing. The character ^ is now a special character in command lines. Including ^200 in a command line will set the associated program's priority to 200. I can't think of any reason this change shouldn't show up on the 6809 soon.

New Module Header Format

Digging into the system a little more we get to the module-header format. There are big changes here with important implications.

There is a field for "System Revision" right after the module sync bytes. Perhaps this will make it possible for Microware to let new-format modules coexist with old-format modules. I wouldn't count on this making a painless upgrade to the new module format possible for 6809 users. It looks more like Microware learned their lesson. This will help with the NEXT migration.

Another interesting addition is the <Module owner>/<Access permissions> combination. I don't think these fields are used yet, but they should make it possible to control the F$SetID SVC. Limiting the F$SetID to the the ID in the main module's header -- and even that only

with permission -- would fill one of OS-9's biggest security loopholes. I've been after Microware to do something about this for a while. I hope they follow through.

One of the standards that has been emerging is for commands to give their "usage," or command line format, on request. One of the new fields in the module header is a pointer to a "usage" string. It's interesting to speculate on why anyone would want to make usage information available as part of the module header. Let me take a very long shot. The usage string associated with the module header could be meant for consumption by another program. It might not even be a printable string. I'd like to see it contain enough information so the shell could use it to prompt interactively for command-line parameters. The result could give use the option of using menu-style command invocation when we choose in the fashion of the Xerox Development Environment or its cousin the Mac environment. Let me emphasize; this is a long and very optimistic shot.

Another pointer in the module header gives an offset for a symbol table. This is a clear sign that Microware is thinking about symbolic debuggers.

Two other fields in the module header are of special interest, Initialized data and Initialized references, but I'll get to those when I talk about processes.

Clues From the New Init

The initialization module has some more interesting clues. Last fall I got excited when I noticed that Microware had documented a way to add new system service requests to Level Two systems (it involved a module called OS9P3). It looks like they've cleaned that up for OS-9 68K. Something called the Customization Module is included in the Init module.

I imagine that many of the academic users of OS-9 have wished for some kind of accounting. I don't think it's here yet, but there's a field for User Accounting in the init module. That should give us hope.

There are two tricks for partitioning processes into high and low priority. Processes are never aged above Max-age (a field in init). That means that a process started with a priority above Max-age will never have to wait for a process started below Max-age. The other trick works from the other end of the priority scale. Processes with a priority below Min-priority are never aged. That means that processes started with a priority below min-priority will only be run when no processes started with a priority above the min-priority threshold are ready to run.

Data Modules

When I use data modules I have to create arrays of junk and turn them into modules with the assembler. That's a silly way to do it. The data modules take up disk space and I/O time and can't be created spontaneously. OS-9 68K includes a new service request that creates a data module on the fly. This fix is easy and useful. I'd be surprised if we don't see it on the 6809 soon.

Process Initialization

A process always inherits the three standard I/O paths of its parent. It would be nice if other paths could be passed as well. Under OS-9 68K the number of paths to pass to the new process is a parameter which must be specified when a new process is started. This is simple change. I'd be certain that it would be on the slate for the 6809 except that F$Fork is already using all the registers available on the 6809. In OS-9 68K they've dumped the Language/type specification when a process is started (no sweat, nobody used it). If they start using register A for number of paths to inherit instead of Language/type, they could give us the feature, but it would make conversion painful. Maybe they could treat all counts above 16 as three. That would make residual Language/type bytes harmless.

There's a much bigger change hidden in the process initialization process. OS-9 has grown something very like a relocating loader. We've always gotten away without a relocating loader by insisting on position independent code and data. C initializers started the slide by permitting a storage initialized with a pointer. That was handled on the 6809

with a combination of a fancy assembler/linker and a special routine that was inserted at the beginning of the program.

Many operating systems include a relocating loader that will initialize pointers when a program is loaded. OS-9 68K has joined the list. During process initialization, fields in a process's data area can be initialized. Two pointers in the module header, Initialized data and Initialized references, give the kernel the information it requires to initialize as much of the process's data storage as necessary. I wonder why they stopped at initializing data. With the full support for relocatable addresses that they must have, Microware could have relaxed the requirement for position independent code.

Libraries

It was beginning to look like Microware was giving up on the module based approach that characterizes OS-9. Well they haven't. Standard services like math functions and (I understand) C I/O functions, are packaged in separate reentrant modules. This is especially clean with a 68000, where service modules can be accessed through traps, but it should be portable to the 6809.

As far as OS-9 68K is concerned this is mostly fact. I'm only reporting what's in the manual. I have made educated (sometimes hopeful) guesses when the manual didn't explain something, or when a feature was latent. As far as the 6809 is concerned, all I have is guess work and Microware's policy statement. Straining their policy of OS-9/OS-9 68K compatibility as far as possible, I think we can hope for:

- some enhanced commands
- Error message file support
- Process priority support in the shell
- Max-Age/Min-Priority support
- A named customization module
- A SVC for Data module creation
- Variable number of paths inherited
- Some library module support

I don't think we'll see extended module headers soon. I'd like to see initialized data areas in OS-9, but I think that would take more space than the Level One kernel can afford. Wild card support in the shell falls into the same category. It's possible, but it takes too much space.

Literature

I hear a lot about the lack of information about OS-9. Well I've done my part. The collection of these columns I put together last summer seems to be clearing whatever snags have been holding it up. I talked with Don Williams about it recently and he assured me that he'll find a way to print it soon. What with the large print I used and the volume of material, the collection ended up over 200 pages long. Don has been trying to find a way to publish it at a price everyone can afford. Maybe I'll have to break down and re-set it in tiny print. Perhaps you'll see it advertised in these pages soon.

That isn't all I did last summer. Dale Puckett and I worked together on a book for Rainbow books. It just came off the presses (in early March).

In my VERY biased opinion its an "outstanding" book. That's not to say I can't find things wrong with it. You know how it is; as soon as it's done you think of a dozen improvements. "The Complete Rainbow Guide to OS-9" is organized more like a book than a manual. It starts with a slow, careful introduction to the use of OS-9 and ends with an explanation of the Level Two memory management service requests. It also includes device drivers (SCF and RBF) printed with Microware's permission. Both Dale and I believe in examples, so the book is packed with little programs, many of them useful.

My family particularly likes the funny little creature that the artist created for the illustrations.

The OS-9 community has gone from famine to feast. Taking the Complete Guide to OS-9 and the collected OS-9 User notes together, there are more than 600 pages of information about OS-9 that are (or will be) available in addition to the Microware Manuals. I should add that I played fair with both of my publishers. The only thing that is duplicated between the two books is the null device driver that I included in my very first column.

I'm sorry if I seem to be tooting my own horn a bit loudly. Let me make it clear that I have an personal interest in these things. If you buy either book I get a slice of what you pay. Please read my description of my work with a suitable number of grains of salt.

"C" User Notes

Edgar M. (Bud) Pass, Ph.D.
1454 Latta Lane
Conyers, Ga 30207

The previous chapter began a tutorial on the C language. The subset presented there was sufficient for writing and understanding many C programs, except that it omitted the standard library functions and some of the advanced syntax concepts, in order to simplify and limit the discussion. This chapter presents most of the remaining syntax of C.

TYPE CONVERSIONS

C compilers allow the declaration of variables and functions and the creation of constants of the following types:

```
char
short (int)
int
long (int)
unsigned
float
double
```

They also support relational and logical expressions of "boolean" type. The C programmer must be aware of the fact that C will silently convert expressions from one type to another, as required, sometimes producing horribly inefficient code, unexpected results, or implementation-dependent interpretations.

This type conversion works so automatically that the user can easily ignore cases in which type conversion is not performed (in many implementations). These cases include function arguments for user-defined and system-supplied functions. For example, the following fragment of a C program will usually fail:

```
isqrt (r)
int r;
{
    return (sqrt (r));
}
```

since the library function "sqrt" expects an argument of type "double" and most implementations do not check argument type on declaration versus call. To avoid this problem, C provides a type conversion unary operator of the following form:

```
(type)
```

which C terms a "cast" or "coersion", and which is used as follows:

```
(type) expression
```

to force "expression" to have type "type". Then, function "isqrt" could be rewritten as follows:

```
isqrt (r)
int r;
{
    return (sqrt((double)r));
}
```

which would change the type of the call parameter "r" of "sqrt" to "double".

Automatic arithmetic conversions generally (though not always) work as expected. When a "lower" type expression is used in the context of a "higher" type expression, its type is usually modified internally in stages toward "higher" types of expressions.

The following sequence of conversions is applied:

- -logical and relational expressions are converted to type "int" values of zero for "false" and one for "true" (although there is some variation in the arithmetic values corresponding to "true" and "false")
- -"char" and "short int" types are converted to "int"
- -"float" type is converted to "double"
- -"int" type is converted to "double" if in a "double" context
- -"double" type is converted to "long" if in a "long" context
- -any type is converted to "unsigned", "int" or "float" or "double" or "long" if in an "unsigned" context
- -otherwise the operation is performed in "int" mode

Hence the warning about the seemingly anamolous evaluation of the following expression:

```
(1/2)
```

as zero was provided earlier: the divide operation is performed in "int" mode, since both of the operands to the divide operator have "int" mode.

A different type of conversion process is applied to assignment expression. The right operand of the "=" operator is evaluated and automatically coerced, if necessary, to the type of the left operand.

The best advice to programmers beginning to use C (and some already using it) is to never allow C to perform automatic type conversion, by explicitly using type casting, or by rearranging expressions involving mixed types. Unfortunately, no known C compiler provides warnings about implicit conversions.

VARIABLE SCOPE

Normally, variables declared outside of all functions are "global" to all functions and those declared inside a function are "local" to that function. However, C allows some exceptions to these rules primarily (although not entirely) to support separately-compiled functions.

If a variable is declared "extern" within a function, the effect is the same as if it were declared outside of all functions. Its existence is permanent for the duration of the execution of the program, and its name is recognized as if it were a "global" from the point of declaration forward.

If a variable is declared "auto" within a function or block, the effect is the same as if the "auto" had not been stated. Sometimes variables are declared "auto" to distinguish them from "extern" variables.

If a variable is declared "static", its name is local to the function in which it is declared, like "automatic" variables, but its value is permanent, like "extern" variables. This enables variables to be declared privately to a function, so that other functions could not disturb their value, and yet retain the previous value upon last exit of the function.

In addition to allowing declarations in functions, C allows declaration in blocks. A block may appear wherever a statement may appear, and is composed of a left brace, one or more declarations, zero or more statements, and a right brace. Declarations in blocks follow the same rules as declarations in functions.

If a function is called before it is declared, the C compiler will not necessarily know that it is a function, since the names of functions are not known before their declaration. Thus C allows the predeclaration of typed functions as if they were variables except that the function name is followed by "()".

VARIABLE INITIALIZATION

Unlike some versions of BASIC and other languages, the standard version of the C language does not initialize "automatic" variables to any particular value when they are created, leaving the user with the responsiblity of assuring correct initialization. It usually initializes "extern" and "static" variables to zero by default.

C provides one convenience to assist in this process in the form of the "initialization" option of variable declaration. When a variable is being declared, it may be followed by an "=" symbol and a value. For example, the following declaration:

```
int i = 2
```

simultaneously declares "i" to be of type "int" and have value "2". More complex forms are possible, such as the following:

```
char string[] = "this is a string"
```

which simultaneously declares "string" to be an array of type "char", length 17, and value "this is a string\0", since C automatically terminates string constants with "\0", which represents hex zero.

The value to which an "automatic" variable is initialized may be any expression which would have been valid had the initializer had instead been an assignment statement; this expression is re-evaluated each time the function is called. Variables of type "extern" and "static" may be initialized only to constants, and will be initialized only once per execution of the program in which they are declared.

To illustrate this point, review the following example function:

```
count()
{
    int i = 0;
    static j = 0;
    i++;
    j++;
    printf ("i = %d, j = %d\n", i, j);
}
```

Note that the value of "i" will always be one at the "printf" function call, whereas the value of "j" will be equal to the num er of times "count" is called.

POINTERS AND ADDRESSES

To a programmer previously familiar only with a high-level language such as BASIC or FORTRAN, the concept of pointers may at first seem foreign and difficult to grasp. However, it is essential for a C programmer to understand pointers in order to write other than the simplest programs.

A pointer in C is a variable which contains the address of another variable. When used properly, pointers are very powerful tools. When used improperly, they can be very destructive and cause otherwise straightforward programs to be virtually impossible to debug or, once finally debugged, to modify.

There are two unary operators in C which deal with pointers and addresses. They are "*" and "&", with interpretation of "fetch contents of address" and "return address of" respectively. Both are highly restrictive in the arguments which they accept. The "&" operator accepts only variables and array elements, not expressions or constants. The "*" operator accepts only "pointer" variables and highly restricted expressions involving pointers and constants.

Pointer variables are declared just as any other variables are declared except for the symbolic "*" preceding the name. Pointer arrays are declared in an analogous fashion.

For example, the following declaration:

```
int i
```

declares "i" to be an integer variable, whereas the following declaration:

```
int *i
```

declares "*i" to be a pointer to an integer variable. Pointer variables must always have an associated type so that the C compiler will be able to handle expressions such as the following:

```
y = *i + 1
```

which sets variable "y" to (the integer value pointed to by "*i") plus one.

It is the responsibility of the programmer to ensure that a pointer variable actually points to valid data of the proper type. The C compiler attempts to flag violations of the rules but cannot usually detect problems such as a pointer pointing past the end of an array, etc.

Since pointers are variables and have types, they may be used with or without the "*" operator to perform various functions. For instance, if "i" and "j" are pointers to integer variables, then the following statement:

```
*i = *j
```

sets the integer variable pointed to by "*i" to the integer variable pointed to by "*j", whereas the following statement:

```
i = j
```

sets the pointer "i" to point to the same integer variable to which "j" points, and does not change either of the "pointed-to" variables.

One common use of pointer variables is to return more than one result from a function without using "globals" or other "side effects". K and R provide the example of the "swap" function which is intended to swap the values of two variables. Since function arguments are always called by value, storing into a simple function parameter has the same effect as storing into a local variable. However, declaring the function arguments as pointers and passing addresses of variables has the desired effect of changing the variables external to the "swap" function.

Following is their example function:

```
DEFINITION          SAMPLE CALL

swap(px,py)         swap(&i,&j)
int *px,*py;
{
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}
```

Note that if all "*" operators were omitted in this "swap" function, only the values of the local parameters, not the values of the parametric variables, would be swapped.

To further relate pointers to the experience those familiar with BASIC, let us define the following expressions:

```
#define poke(adr,n) (*(char *)(adr))=(n)
#define peek(adr)   (*(char *)(adr))
```

which modify and inspect memory location "adr" respectively, just as do the BASIC "POKE" statement and "PEEK" function. Note the great symbolic similarity between the two definition bodies.

ARRAYS

In the first chapter of this tutorial, the definition of an array was said to be the same as the definition of a variable except for the addition of a constant enclosed in brackets after the variable name. Later this was modified to include the case of initializers in which the constant may be deleted, but the length of the initialization string provides the missing length. The length may also be omitted for function parameters which are one-dimensional arrays. Multi-dimensional arrays use the following type of notation:

```
int arr [10][20]
```

rather than the more familiar notation used in languages such as BASIC and FORTRAN in which commas separate the dimensions.

What may be surprising (at first) to non-C programmers is that pointers and arrays of the same type are highly related and almost (though not entirely) interchangeable.

Given the following example declaration:

```
int arr[100],*ptr,i;
```

then the following assignments:

```
ptr = &arr[0];
   and
ptr = arr;
```

both point "ptr" to the first element of "arr", and the following expressions:

```
   arr[i]
    and
*(ptr + i)
    and
*(arr + i)
    and
*(&arr[i])
```

all refer to the "i-th" element of "arr", since the C compiler interprets the following expression:

```
(ptr + i)
```

such that the "i" is multiplied by the length of one occurrence of the variable type associated with "ptr" or "arr".

However, the pointer expression syntax is quite restrictive, in that expressions of the following forms:

```
(ptr + ptr)
   or
(ptr * ptr)
   or
(i + ptr)
```

are illegal or meaningless, at least in standard C.

An array name always points to a preallocated storage area, as opposed to a pointer, which does not. Thus, the C compiler allows expressions such as the following:

```
ptr = arr
   or
ptr++
```

but disallows expressions such as the following:

```
arr = ptr
   or
arr++
```

since this would cause "arr" to no longer point to its original storage area.

The decision to process data by the use of arrays or by pointers is made by the user in terms of which is more convenient to the situation. Experienced C users will often use both concepts in some fashion.

POINTERS VS. ARRAYS

A pointer in C is a variable which contains the address of another variable. An array in C is a replicated variable. However, C allows an array name without the subscript to represent the address of the beginning of the storage allocated to the replicated variable. A pointer may have its value changed to point to any desired address. An array name may not have its value changed, since it would no longer point to the original address.

Consider the complex cases, possibly involving pointers, arrays, and formal parameters. C compilers and humans may become quickly confused by the potential combinations of operators and operands.

Consider the two following declarations of functions:

```
fcn(a)
char *a[];
{ ... }
    and
fcn(a)
char **a;
{ ... }
```

where "..." represents the body of the function.

These two will produce exactly the same code, but the first is technically questionable. The first case declares "a" to be the address of a block of memory used as type char; the second declares "a" to be a pointer-width object which will be used to contain an address of a block of memory used as type char. The technical difference is one of degree of dereferencing. That is, one symbol is bound to an absolute address of a group of characters, and the other is bound to a location which holds the address of a group of characters.

Most C compilers will accept the declaration "char a[]" as a formal parameter and treat it exactly as they would treat "char *a". In particular, many C programs declare the "main" function as follows:

```
main(argc,argv)
int argc;
char *argv[];
{
    :
    :
}
```

However, this is a special case applicable only to format function parameters, and which will be discussed at some length later in this chapter. In all other cases, the two declarations will produce different results. In particular, if a UNIX C program references the standard error messages, the following declaration:

```
extern char *sys_errlist[];
```

will produce correct code and the following declaration:

```
extern char **sys_errlist;
```

will not work correctly.

There is only one case in which using the empty bracket notation on function formal parameters instead of pointer notation is necessary. This is in the case of passing multi-dimensioned arrays.

Consider the following example:

```
char text[250][512];
func (text);
    :
    :
func (array)
char array[][512];
{
    :
    :
}
```

In this case, the declaration should get correct treatment for indexing into the argument array, even though the symbol "array" is bound to a location which contains the address of the array "text" in the calling procedure. This will work for any number of dimensions, but the (n-1) outer-most sizes must be stated at compile time. This limits the usefulness of multi-dimensioned arrays somewhat as the dimensions of the array may not be passed as arguments to a function.

Local function variables follow the same rules for externals, with the exception that the default storage class is automatic instead of external, and that aggregates (arrays and structures) may not have initializers. Technically, automatics are not initializable. Code is generated to set the initial values each time the function is called as part of the function entry process.

In this context, it is unnecessary to defer space binding until link time, because the space is created when the function is called. The compiler must determine how much space to reserve to generate the proper code. That many C compilers accept empty brackets for an automatic array is probably an error derived from the processing of the special case of function parameters.

The point of this discussion of pointers and arrays is that the differences are not always clear, and that even the best C programmers and compilers are sometimes mislead by the symbology used in the C syntax. When in doubt, keep it simple.

COMMAND LINE ARGUMENTS

Since C is heavily used in systems programming, it has several features not often found in general-purpose application languages. One of the primary special-purpose features is the ability of C programs to directly access the command

line used to execute the C program in order to retrieve arguments potentially placed there by the user.

When the "main" function is called, it is always called with two arguments, generally called "argc" and "argv", represented symbolically as follows:

```
main (argc,argv)
int argc;
char *argv[];
{
    :
    :
}
```

in which "argc" is the number of parameters and "*argv" is a pointer array referencing the parameters. Since "*argv[0]" is the name of the program, "argc" will always be positive. K & R provides the following example of a trivial program which echoes its argument:

```
main (argc,argv)
int argc;
char *argv[];
{
    while (--argc > 0)
        printf ((argc > 1) ? "%s" : "%s\n",
            *++argv);
}
```

STRUCTURES

Structures are often used in C programs to help the programmer organize data structures into units rather than separate entities. They have no direct analogy in BASIC or FORTRAN, but are similar to PASCAL or COBOL record formats. The closest BASIC concept is the FIELD statement.

Unfortunately, their implementation varies widely among the versions of the C compiler, and most implementations place severe restrictions on the use of structures, such as not allowing them to be assigned or passed to a function as a unit. Because of these restrictions, almost all structure manipulation is done using pointers.

A structural template is declared as in the following example:

```
struct birthinfo
{
    int year;
    int month;
    int day;
    char name[20];
}
```

which reserves no storage but defines "birthinfo" as a structure containing three integers ("year", "month", "day") and one character array "name" of twenty characters. The template may be used directly, by following the closing brace of the structure definition with one or more variable names, separated by commas. It may also be used indirectly, by following the word "struct" and a previously-defined structure name with one or more variable names, separated by commas. Structured arrays and pointers to structures are declared in a fashion analogous to the definition of integer arrays and pointers to integers.

A structured unit may be declared and initialized in a manner similar to that used in the following example:

```
struct birthdate person =
    {1952,4,25,"John Doe"};
```

A member of a structured unit is referenced as follows:

```
structure-name.member-name
```

or for example, the following expression:

```
person.name
```

would represent the "name" field of the "person" structure.

Structures may be nested but may not be recursive, although they may contain pointers to structures which could point to themselves, in some cases.

The use of pointers to structures is so common that the following notation is provided by many C compilers:

```
p -> member
```

to represent the following expression:

```
(*p).member
```

assuming "p" represents a pointer to a structure and "member" represents a member field of that structure.

UNIONS

Whereas a "struct" declaration represents a record format or sequential data structure, a "union" declaration represents a shared data structure or "either-or" format, somewhat similar to EQUIVALENCE statements in FORTRAN and FIELD statements in BASIC.

The syntax and usage of a "union" and of a "struct" are identical. The template represented by a "union" is made large enough by the C compiler to hold the largest member. For example, given the following "union" declaration:

```
union CIO
{
    int AFL;
    float NFL;
    char *WHL;
}   TEAM;
```

then the following variables would all start at the same memory address:

```
TEAM.AFL
TEAM.NFL
TEAM.*WHL
```

even though they are of different types and lengths. The C programmer must ensure that the correct data is available when needed.

SUMMARY

This chapter continued the tutorial begun in the previous chapter. If you have access to a C compiler, use it. If you do not have a copy of "The C Programming Language" by Kernighan and Ritchie, buy it. Enter and run several of the programs in K and R.

68000 User Notes

Phillip Lucido
2320 Saratoga Drive
Sharpsville, Pa 16150

Odds and Ends

Things have moved rather slowly here over the past month. Perhaps it's just that winter is dragging on, refusing to give way to spring, but everything seems to have quieted down. I am not entirely without topics for this month's column, though. Even in the slowest month, there's always something happening to report on.

I Wanna Hack!

For instance, I did just buy a brand new computer. I've had my new Macintosh for a little under a month now. You'd think that after that much time I'd have plenty to say about it, but unfortunately that isn't so.
You see, I have a confession to make. I am not really a computer user. I'm a computer programmer and hacker, and there can be a big difference. After all these years, I'm still not entirely sure just what people who don't program actually do with their machines. I have very few 'practical' uses for my computer. In fact, about the only thing I use it for which doesn't have to do with programming is writing these columns. This dedication to programming makes me part of a very small group, which happily includes many of the people who read this magazine.
The Macintosh is advertised as the computer "for the rest of us". If that is so, I must be one of "them", as opposed to "us". If I can't get into a machine, at least in the software sense, I don't have a lot of use for it. And as sold, a Macintosh is not oriented toward the programmer.
From what I can see, the Macintosh is a software marvel. The problem is that I just can't see very much. I've heard that the Mac has 64K of ROM, with hundreds of entry points to make programming in the window environment possible. Don't expect to see any of this in the manuals that come with the Mac, though. They are definitely written with users, not hackers, in mind.
So what is the despairing hacker to do? Well, Apple has released all of the necessary information. There is a three volume set, called **Inside Macintosh**, available from Apple for $100. The books are supposed to run to about 800 pages of densely packed info. I, of course, ordered them soon after purchasing the Mac. That's well over three weeks ago. They haven't shown up yet. Do you have any idea how frustrating it is to have this glorious new machine, and not be able to hack?
Oh, well. Whenever the books do arrive, I'll be able to dig in and get to work. In the meantime, I guess I'll just have to be satisfied with drawing pretty pictures. Actually, I do have a C compiler already. I bought the Aztec C compiler from Manx, purchasing the flow-blown commercial system with full library source code. While this does offer tantalizing glimpses into the Mac, the Aztec C documentation does read, in effect, "now pull up a chair with this book and your copy of **Inside Macintosh**, and get to work".
Aztec C is really quite nice. It replaces the desktop/icon/mouse environment, for purposes of program development, with a Unix-type shell, with I/O direction, script files, and hierarchical directories. It also supplies a programming editor, called 'z', which runs a subset of the standard Unix editor 'vi'. Since I use Unix and vi at work, there was very little delay before I could start programming. I can now write all the Unix-type utilities in C to run on the Mac under the Aztec shell that I need. That's not what I'm interested in doing, though. Where is my copy of IM, Apple? (Sorry for the whining - this really is frustrating!)

Mac Impressions

I'll have more to say about programming the Mac in C when I finally do get all the information I need. Meanwhile, there are a few things I can say, in my (admittedly limited) computer user persona.

First off, MacPaint is FUN. I may not have much need for it, but it is a great way to waste hours of time. I've already turned out a decidely unflattering portrait of the president of the local CoCo users group, which went into the club newsletter. There have also been some rather strange abstracts popping up around the house. I'm not sure what they are, but they're fun to make.

The other piece of bundled software is MacWrite, the word processor. I haven't done any more than glance at this one yet. It just seems to be too slow when compared to what I can do with my OS-9 system. I probably will check it out in the next week or so, since something I'm doing right now could really use the 'font' abilities of MacWrite, especially the ability to produce text using all of the strange mathematical symbols like summation and integrals.

In addition to the C compiler, I have purchased one program so far. As soon as I bought the Mac, I also bought a copy of Sargon III, the chess program from Hayden. I figured that with the power and speed of the 68000, I might get a fairly decent game of chess from the computer. Uhhh - right. So far, the only way I can beat it is to play it at the lowest possible level, where it takes only 5 seconds a move and doesn't think on my time. I knew I was rusty after laying off active chess playing for six or seven years, but this is ridiculous!

On top of the bundled and purchased software, I also have acquired a number disks packed with public domain programs. The Macintosh has a thriving programming community, centered around MAUG (Micronet Apple User's Group) on Compuserve. These programmers are turning out large numbers of useful applications, both programming utilities and more mundane things like games. The utilities will probably be most useful once I get programming myself, though some of them are useful to the general user. The games are quite good, and not just because they are free. For instance, I have no less than three versions of Life, the John Conway cellular simulation game, all of which simply fly when compared to 8-bit micro versions that I have seen.

So much for the software I have. There are also a number of things I can say about the Macintosh in general. First, the Mac is physically a marvel. It takes up very little space on a desk (it has a very small 'footprint', as the terminology goes), though the mouse requires me to keep some empty space on my desk, which is usually piled with junk to a sufficient depth for geological strata to form. The screen on the Mac is also very nice. Though it might seem too small, it is actually quite crisp and easy to view. Finally, the floppy disks, encased in their hard plastic shells, are convenient.

Now for the gripes. The desktop and icon environment, as provided by the program called the 'Finder', is useful and easy to use, but it is too slow. Starting a program from within the Finder usually takes a minimum of 15 seconds, which is an eternity when compared to the hard disk response I am used to. The problem is not all due to floppy disks versus hard disks, either, but is due to the Finder itself. Also, when an application is completed, the Finder is reloaded and all disks currently in drives are reread, again taking forever. Another problem has to do with the storage capacity of the floppy disks. These are single sided disks which store 400K apiece. This might be sufficient, except that any disk which is bootable must contain the various system information, which right off the bat consumes almost half a disk. While it is not necessary to make every disk bootable, it is often much easier to do so just so the Mac is not constantly ejecting disks and prompting you to put the boot disk back in the drive so some part of the system code can be accessed. While double sided disks are supposed to be available in the future, no one is saying just when that will happen.

As a final gripe, I don't like having to deal with copy protection schemes. One nice thing about OS-9 has been the absence of any screwy protection schemes, which are meant to prevent piracy but instead just make software harder to use. Such barbarities seem to be reserved for the truly huge software markets, for which the Macintosh certainly qualifies. The chess program cannot be backed up (though I haven't tried yet), and a backup is only available by sending $10 to Hayden. This for a program which cost me only $50, and is available for as low as $33 through the mail. Phooey! The C compiler is a little better. Of the three disks on which the compiler is supplied, two have a software key which is not transferred by the normal Finder disk-to-disk copy. You can copy these disks, but when you boot from a copy, the Mac will eject your disk and ask for the original disk, to verify that you are a proper user. It will then eject the original, and you can continue from that point on using only the copy.

Actually, both of these protection schemes are easily breakable. There are several bit-copy programs which will make image backups of disks, no matter what protection scheme is used. There are also methods for those who know a little more about the internal structure of Macintosh disks. With all of this, why protect in the first place?

That's it for the Mac, at least for this month. More later on once I get the info I so desperately crave.

Anybody for a Megabyte?

My current bus machine, the Hazelwood Helix, has 256K of dynamic RAM. When I first got it a year ago, I figured that this would hold me for a long time. After all, it had taken me years just to work up to 64K, hadn't it? It has become increasingly clear, though, that lots and lots of RAM is a good thing. With enough RAM, I can preload just about every program at startup, so that the delay caused by computing the OS-9 checksum must only be endured then, instead of every time I use a program. Also, enough RAM allows me to allocate a large RAM disk, so temporary files used by compilers and the like can be read and written at RAM speeds. If I want to preload all of the C compiler phases, though, and still leave enough for a RAM disk, 256K is just not enough.

All well and good, except the 256K by 1 bit dynamic RAMs that I would need to upgrade to a megabyte were just too expensive. Too expensive, that is, until the past few weeks. In a rather astounding example of price wars and supply catching up to demand, the prices of the 41256-type DRAMs have tumbled. In the latest computer magazines I have, ads typically quote prices of $15 to $25 or more per chip. Try calling a chip dealer, though, and see what you get! I have just sent off for 32 of the RAM chips, at a cost of $6.25 per chip! For just $200, I can upgrade my 256K RAM to 1 megabyte. If I waited another week or two, even that would probably drop another 50 cents per chip.

There are still some problems. The board I am upgrading is a Hazelwood DM-256, which does not have any provision for replacing the 4164s with 41256s. I therefore am getting ready to perform a little hardware hacking (and slashing), something I haven't done for a while now. If you have the same board and would like to know how to perform the modifications, drop me a line. I'll probably be writing up the procedure anyway for the mag. Anyway, hopefully by the time you see this column, I'll be running with a full 1016K of RAM (8K reserved for I/O and boot ROM).

Ramblings & Such

Every now and then something, or somethings happen that leads me to indulge myself with these ramblings and such. Or as sometimes happens I just get the feeling that there are some things that need discussing, between you and me. This is one of those discussion. First, I tell you what I am thinking, or know, and then I wait for your reply. Every time I have embarked on one of these get togethers I am gratified that you, or at least many of you, take the time to reply. That way I sorta keep up with what you are doing and thinking about those subjects we cover in 68 Micro Journal. Then we plan ahead taking your replies into serious consideration. Fact is, your input has had more control over our direction, than the thoughts or desires of any of our staff, and that includes me. Sure, I am always hearing from some of you who want more FLEX, and others wanting more OS-9, and others wanting more CoCo. Others want more 6800, 68000, or more of some other subject. That is what happens when we have a diverse readership. We try, really try to spread it out so that no one gets left out. It is a tough decision. Think, how would you handle it?

When we first started, back over seven years ago, I determined then that this was to be a publication for the readers. The advertisers were to be secondary. It is reasonable to assume that if we HAVE and KEEP readers, then the advertiser benefits. It is still that way today. Not that it has been the best method of operation from the financial aspect. Actually we have backed ourselves into a corner. We are the most restrictive computer magazine in the business when it comes to requiring the advertiser to prove the truth of his advertising. **We lost a lot of income by refusing and/or dropping advertisers who did not, or would not, for some reason or another, comply with our restrictive advertising standards!** But then it all

went back to that original decision, **THE READER WOULD COME FIRST!** Look at how many have been cheated or mislead in advertising in some of the other publications, in the past. A couple have gotten by us, and a few slipped after they started advertising in 68 Micro Journal, but we stepped in, when we found out for certain. We let you know also. No punches were held back. We lost advertising revenue, but we kept the faith! The next time you buy a computer product, remember, our advertisers supported this policy also, and YOU owe them something for that honesty, YOUR support.

Many of you were probably not around when we first started. So, for you a little history. I, and many other 6800 users back then were getting pretty deep into the S50 bus and 6800 computers in general. At that time we were the second largest group of computer users in the world. The S100 bus, now defunct for all practical purposes, was the first by a month or so and the largest in number of systems and users, but only by a small margin, in those days. Of course there were other brands of computers but they were much larger in scope and actually not grouped, as we in the microcomputer field were.

Fact is, there was more pages of published material for microcomputers in their first year, than all the published pages, for all the bigger (IBM, DEC, Data General, Wang, etc.) at that time. And they had been around for many years. However, one important (to us 6800 users) thorn existed, we didn't get very much attention in the early publications, of which there were few.

Actually we sometimes went for months without finding anything, in any of them, that pertained to us as 6800 users. And as time passed, the drought deepened until we were forgotten, for all practical purposes. Despite the efforts of early authors, such as Peter Stark, Mickey Ferguson, Dr. Levy and even maybe myself, and a few others (my apologies to you who I have left out, it is 4:30 a.m. and the old noodle is a bit draggy.) Yes, we sold articles, but the tide was turned to the Intel group and S100, despite the booming truth that the 6800 was far, far superior to the then cream of the Intel bunch - the 8008 and 8080 CPUs. And they say, "Maytag repairmen get lonesome", boy!

I became livid at times when I had sold an article and then it did not make it to print because, we were not source supported as well as the Intel group was, and our suppliers did not spend the advertising bucks theirs did, and after all if they paid for the article, we could not ask for it back and the other mags then did not get a shot at it. I finally wised up!

I was pretty fortunate at the time as I had become sorta ho-hum about being a newspaper publisher, having semi-retired from the electronic design and support game, a few years earlier, and purchased a major share of a newspaper after returning here to the Hixson Tennessee area.

I acquired one of the early 6800s and started into it, along with another local hacker, Mickey Ferguson, and his computer wise wife, Foxxy. I was boot-strapping myself into the digital world from those tubes and such and Mickey was into the thing like a power drill into jello. I learned a lot from him (especially the vector (index) register) and later even wrote a complete program using only the index register of the 6800. Old dogs learn slower. But we manage.

Also my family operates one of the largest typesetting and color separation facilities in the South East. That along with my newly acquired newspaper experience started me to think - "If they (other mags) are going to ignore us, then I would start a news-letter, for 6800 users." A simple little thing, or so I thought, just to exchange ideas. Heck, I even had the paper and presses, and a second class mailing permit. All I needed was some input (articles, hints, kinks, etc.) and we would be off and running. No thought of advertisers, at first, nor even once a month. See how much I knew about the magazine business.

After some discussion with Mickey about the project I began to feel that I could get away from the boredom of newspaper work and maybe break even on putting stuff out about my newly acquired hobby. In the world of magazines I had not even progressed to the kindergarten stage, but was soon to find out.

While kicking the thing around with Mickey on a local 2 meter (ham radio)

repeater one morning about 2 or 3 a.m., a voice 'broke' into the conversation and said that he also was interested in computers and the 6800 in particular. Well, he gave his call and Mickey immediately knew who it was (I still marvel at the extent of recall Mickey has) - it was Peter Stark of Kilobaud fame. Boy, a real live celebrity, and one who I had followed each month with anticipation. For at that time Peter was about the only 6800 author left that was getting any play in the magazines. He was a regular in Kilobaud and well respected in the magazine as well as the 6800 community. The year was 1978.

After exchanging a few remarks, and Peter telling us what hotel he was staying at in Chattanooga (8 miles for me - 22 for Mickey) Mickey and I drove down and picked Peter up for a early breakfast, at one of the local ham and egg establishment. There we told him about my plans for a news-letter. After finishing our meal we all piled into my car and drove out to my place where everything was set up in a back bed room. At that point I would have given my prom burner to get Peter to write for us, but true to his nature, he explained that he had an implied loyalty to Kilobaud and Wayne Green, the publisher, and I had no desire to have him breach the faith Wayne had placed in him. It was over three years or so before Peter ever sent in anything for us to publish and I respect him to this day for it. However, Peter - keep it coming now!

Also I had one each of everything that SWTPC had made, and a couple of things that were not yet released. Peter was using primarily the Percom system and as I had it also we swapped a few pointers (mostly from him to me, I couldn't tell him much) and also talked about my news-letter. His encouragement, despite the fact he could not participate, went a long way in firming later my decision to continue. After returning him to his hotel about 6 in the morning, Mickey and I went home. Mickey who worked nights was used to late hours, I had enough help at the office to sack in late, but Peter who had to be out to our local university that morning at about 8, got no sleep. Sorry Pete!

Now determined more than ever to start the thing for 6800 users I dropped a letter to a few friends around the country, who I felt could give some input and had a good knowledge of the 6800 and its software needs. Well, word got to Dan Meyer of SWTPC and he immediately called me and suggested that if I was interested he (SWTPC) would take a page or so. He then told Dave Shirk of TSC and Dave called also and offered support in advertising and software. With this sudden influx of advertising funding promised I then send letters to all manufacturers of 6800 products and became a fledgling magazine before the first issue was printed or mailed. In the black the first month! Boy!

Actually I was pretty flabbergasted as Wayne Green, an old friend, had suggested in a letter a few weeks previously, that the market would not support a magazine for the small group of 6800 users. I respected Waynes comments as he was deep into the thing, having started Byte and later Kilobaud (not to mention his years of experience at CQ magazine (ham radio) and later his own 73 Magazine, which both Mickey and I had sold articles to. Also it was 73 where I first remembered Peter and his 6800 stuff. Yet, in my ignorant bliss I jumped in; and here we are into our 8th year. Just goes to show how it is hard to call shots in the micro industry. As Mickey used to say - "nobody told the bumble-bee that he didn't have the right stuff to fly, so he just goes ahead and does it."

As I look back I believe the reason we lasted is because we put the reader first. It cost a lot in lost revenue from advertising rejected or stopped, but then it is better to have not made it real big but survived, than to have made a big splash and then sank. As it is now, we can go on a long time, even with the present situation of a retracting market, because we have modeled our operation, and it's support divisions (Data-Comp - S.E. MEdia, etc) to the point that we can look ahead and adjust. Also it is **ONLY BECAUSE OF THOSE SUPPORT DIVISION THAT 68 MICRO JOURNAL EVEN SURVIVED ONE YEAR.** We see what can happen, and even is happening but we are prepared. We have our own printing facilities, own our own buildings and combine the revenues of all operations into a common source of support funding. As a group, **AND WITH YOUR CONTINUED SUPPORT** 68 Micro Journal will be around a long time. Actually as long as you are out there and want 'your' magazine. You keep the good

articles coming, support our advertisers and I will do the rest. We might have to go to a less quality cover or inside paper, or undertake other cost saving moves, if the situation warrants, but we **WILL HANG IN THERE!!!!!!!**

Which brings me to the point that prompted me to start this discussion with you. I wanted to assure you that we (CPI) are dedicated to keeping 68 Micro Journal the 'only' real 68XX magazine. Advertisers who have from time to time advertised in other 'start-up' 68XX magazines (who have all dropped by the way-side) tell me that ONLY in 68 Micro Journal do they get the kind of results needed to insure their business. That alone keeps us going. For as you should know, your subscription costs do not completely cover the production and mailing cost. Yes, I know that a lot of other magazines have more pages and prettier art work than we do, but they have many times over more users and readers than we do. And that is what makes the difference, the unit cost depends on volume. If our numbers could grow, we would have more readers, more advertisers, less cost per unit and all of us would be happier. But, as it is, we have enough to make a real go of it, as long as you support us, as you have in the past.

You would be surprised as to how many letters we get directed to "Dear Club Members', or something similar. I guess we are really more a clannish or club like group, than a full blown commercial, big time magazine. And frankly, I like it better this way. I have made more lasting friends and enjoyed my work more, since I started 68 Micro Journal, than any other span in my life. And I owe it all to you, the readers and advertisers, who supported us ALL. Right, not just me, but **ALL OF US.**

And together, and I really mean **together**, we have made this thing work, when the experts were telling me it was impossible. Your support of 68 Micro Journal, Data-Comp and S.E. Media, allows me to feel confident that we can meet the changing times! For you see, that is why some of the other magazines never made it in the 68XX field. Not that the folks running them were not up to snuff. Quite the contrary, some, actually most, were far more knowledgeable than I am about computers and such. And decent sorts of chaps. They had prettier art work, some had better make-up and lay-out, some used better paper and other aspects of magazine work up that is required in most cases. However, they could not understand that to make it required more. It required material (articles) that the readers wanted. It demands reader participation and support. It requires that we attempt to insure that they will not be ripped-off by dishonest advertisers and much more. In our case the much more being, in part, we have other supporting divisions. So, now I tell you a proven truth -**NO MAGAZINE CAN SURVIVE, AND SUPPORT YOU HONESTLY, WITHOUT OUTSIDE FUNDING, IN A GROUP AS SMALL AS OURS IS!** And that includes 68 Micro Journal, from day one. Our support comes from our support divisions and funds laid aside 'for a rainy day'. If our group was a real money deal, you can bet your CPU the big boys would have been here a long time ago, chewing us up.

In the months to come you will see a thinning out of advertisers in 68 Micro Journal. Actually most of the other magazines are seeing the same thing. Most will be leaving still kicking, but directing their efforts to other markets not covered by 68 Micro Journal. To them we can only say a heart-felt thanks for the support and fine products they gave us, and wish them all the best in their new direction.

One or two may fallout because of other reasons (reasons to be released at a later date, as facts confirmed) but that has been happening since we first started. We (you and I) run 68 Micro Journal and determine the standards for advertising, and if they cannot conform, as the others have done for years, then we bid them good luck, as they ply their wares elsewhere. For I stand firm to the commitment I made in that first issue to demand advertising integrity.

Therefore, I want to let you know now that we are here to stay. We will have more advertisers, after the shake-out, than we had in the first issue, and it was in the black. **AND I OWE IT ALL TO YOU - OUR LOYAL READERS - YOUR SUPPORT OF US AND OUR ADVERTISERS HAVE MADE IT SO, AND WILL CONTINUE AS LONG AS YOU WANT!**

DMW

ADA® And The 68000

Theodore F. Elbert
The University of West Florida
Pensacola, Fl 32514

Part 2 Characteristics of the ADA Language

Ada is a modern, high-order programming language that was developed under the sponsorship of the United States Department of Defense for one specific purpose: to reduce the life-cycle software costs for embedded computer systems. Embedded computer systems are usually defined to be those computer systems that constitute a part of a larger system whose primary function is other than computational. In the case of the Department of Defense, the application is clearly to weapon systems and other military uses, but the general concept of an embedded computer system applies equally well to process control, communications systems, and to many other non-military uses of computers. In fact, the development of Ada has produced a superior, state-of-the-art, general-purpose programming language that incorporates most of the modern principles of software engineering. Indeed, many of the features of Ada may seem strange and even cumbersome to an experienced programmer who is not familiar with modern software engineering principles such as information hiding, abstraction, modularity, and localization, among others. Modern software design methodologies, such as structured programming, top-down design, and object oriented design, are fully supported by the characteristics of the language.

While the development of Ada was directed specifically toward the embedded computer system, the resulting language is well suited to other applications as well. Other areas, such as general systems programming, industrial process control applications requiring real-time concurrent processing, general applications programming, and scientific computation, can all benefit from the modern features of Ada. In the educational field, Ada has been adopted by some institutions as the primary language of instruction because its features force the user to apply good programming practice.

Ada is a modern algorithmic language with control structures similar to those of other block structured modern languages such as Pascal. Ada also provides the user with the ability to define his own data types and subprograms, in a manner analogous to that found in Pascal. It provides for modularity by a unique feature called a package, in which data, data types, subprograms, and other packages can be encapsulated. Modularity is supported in a physical sense as well by the separate compilation facilities of the language. Special real-time programming features of the language include concurrently executing threads of code called tasks, and the ability to handle

exceptions. Synchronization of and communications among concurrently executing tasks -- as well as exception handling -- are provided within the language itself, making it unnecessary for the program to effect calls to the underlying operating system in order to provide such features. Also provided within the language itself is precise control over the representation of data in the underlying hardware, and access to system-dependent features. Finally, the generic facilities of Ada permit parameterization of subprograms and packages, a process by which a template of the subprogram or package is compiled -- lacking some essential features which are provided later as parameters in a process called instantiation. A given generic unit can be instantiated any number of times in an Ada program, using a different set of parameters for each instantiation.

Ada is a strongly typed language, meaning that data objects of a given type may be assigned only those values appropriate to the type. Furthermore, only certain predefined operations may be performed on data of a given type. Type checking is performed at compile time, thus providing early detection of errors associated with data typing. A unique Ada data type -- the private type -- provides the programmer with the ability to create abstract data types. Data abstraction is a modern software engineering concept by which details of an implementation are hidden from the user, while at the same time mechanisms for the use of the implementation are provided. In this manner, the software equivalent of a "black-box" can be created.

It is important to be aware of the design goals of the language developers, because many of Ada's most visible features may seem cumbersome or even counterproductive to those not aware of the intended application domain, or to those unfamiliar with the more modern concepts of software engineering. The application domain was that of the Department of Defense embedded computer system. In general, embedded computer systems have definite characteristics, as listed below:

- Programs tend to be large, on the order of tens of thousands or even hundreds of thousands, of lines of code.
- They are in service for extended periods, perhaps up to twenty years.
- They must be fault-tolerant, recovering from faults or gracefully degrading to a lower level of performance. This requirement necessitates software response to exceptions in the processing.
- They undergo continuous field changes, including software updates due to design improvements or changing operational requirements.
- They must have high reliability, both in hardware and in software.
- There are usually physical constraints in terms of hardware size or processing speed, implying a requirement for efficiency in the software.
- Input-output requirements are usually specialized.
- There is usually a requirement for concurrent processing.
- There is usually a requirement for real-time processing, in that the system must respond to physical stimuli in real-time.
- There is normally a host-target relationship, with software development taking place on a large host computer equipped with software development tools, while the actual application program executes on the embedded target computer containing only those features required for mission functionality.

This particular domain of embedded systems was targeted by the Department of Defense because the preponderance of software costs were incurred in programming for embedded systems, and because software life-cycle costs were rapidly becoming by far the major contributor to overall system costs. Furthermore, software maintenance costs were exceeding those of initial software development, sometimes by an order of magnitude or more. Thus, the potential was there for an enormous savings in system life-cycle costs if a programming language could be developed that would provide features specifically oriented to life-cycle cost reduction. The reader should note at this point that minimization of life-cycle costs may well produce initial development costs well above minimum, because of maintenance enhancing features incorporated into the software design.

To accomplish Ada's design goal of minimizing life-cycle software costs of Department of Defense embedded computer systems, the language was developed with three primary concerns:

- reliability and maintainability of the resulting software
- efficiency in terms of execution time and machine resources
- the realization that programming is a human activity, and that the language features must take into account human limitations.

These concerns were fundamental to the design of the language; the reader should keep this in mind as the various language features are explored later in this series of articles.

Because of the emphasis placed on reliablity and maintainability in the development of Ada, it is not surprising that the language promotes program readability -- often at the expense of ease of writing. A properly written Ada program is almost self-documenting, since the syntax rules permit English-like constructs. The strong typing features of Ada enhance program reliability, and the incorporation of exception handlers into the language enhance run-time reliability. The separate compilation features of Ada also support the maintainability of programs written in the language.

The efficiency of the language in terms of execution time and machine resources was a key concern during the development of the language. Although Ada contains several features not found in other languages, none of these were considered to be exhorbitant in terms of machine resources or of execution time in the run-time environment.

As a human activity, the programming of large systems presents a perplexing problem -- the management of the complexity of the resulting software. Indeed, this single feature of the modern programming environment has been identified as the major contributor to the production of poor software. Many features of good software engineering practice -- such as abstraction, information hiding, and modularity -- have as one of their purposes the management of complexity. The Ada language design incorporates these and other features which tend to reduce the complexity of the resulting software. Ada also provides features, such as separate compilation, that provide the ability to assemble a program from independently produced software components. The independent production of software components is characteristic of the development of large programs in the embedded system environment.

In general, the term software engineering implies a disciplined approach to software development that requires the application of principles completely analogous to long standing practice in engineering design. This process, in turn, calls for the delineation of a set of operational requirements, together with the specification of general characteristics of the resulting software. While the operational requirements are specific to the particular software project, the general characteristics to which all software development should aspire have been detailed. These general chatacteristics, which -- when adopted -- become general design goals, are:

- modifiability
- efficiency
- reliability
- understandability

In addition to these general design properties, the Department of Defense embedded system should exhibit the properties of:

- portability -- both of software and programmers
- reusability.

The Ada language design was driven by the need to provide these general design goals in Ada software. This need, in turn, was met by incorporating into the language the means of applying the software engineering principles of abstraction, information hiding, modularization and localization. The resulting language features -- strong typing, separate compilation and others -- all support the attainment of these general design goals in software developed through use of the Ada language. Each of the general design goals -- modifiability, reliability, efficiency, understandability, portability, and reuseability -- and the means by which they are supported by the language will be treated in Part 3 of this series of articles.

NEXT: Software Engineering Aspects of the Ada Language.

Ada[R] is a registered trademark of the U.S. Government (Ada Joint Program Office).

Basic OS-9

Ron Voigts

WHISTLES, BELLS, and FLASHING LIGHTS

Someone once said, "Never judge a book by its cover." Evidently he wasn't in the business of selling books. Last time I went to a bookstore, all the 'best selling' books and magazines had flashy covers, that jumped up and said, "look at me!" What you run on your computer is the same. No matter how good your program is, it won't help if you don't keep your user's interest. How your program works is important, but if no one looks at it, forget it! Programs that make it are the ones that jump up and say, "look at me!"

Some years ago I wrote a rather large program to interface some test equipment to our main computer. The program worked well, but didn't really get your attention. It needed something to make it more noticeable. The computer I was working on used an ADM-3A terminal. This was definitely not the top of the line CRT, but it did have a few simple control characters. One would home the cursor, another clear the screen and still another made it go "beep". I rewrote the program. When I was finished, the program would start by clearing the screen. Next it would print a menu and prompt the user for a response. If the wrong selection was entered, it would beep and flash an error message.

As time went on and the terminals became more sophisticated, the special effects became better. There was blanking the cursor, moving it around, creating graphics, and more. Other programmers came and asked how to add "whistles, bells, and flashing lights" to their programs. Some of the programming languages were easy to work with while others presented a challenge. One of the easiest systems to use overall, I found, was OS-9.

Whether you are at the "System" level, or working from some language, it is easy to move control characters to your terminal. There is a consistency throughout the entire system that makes a technique in one language transferable to another. Now I have to mention that many of the examples I use apply to the standard Color Computer screen. Many of the effects can easily be applied to another terminal. In fact, many transfer directly to other terminals. The important thing is that you want to cause OS-9 to "echo" a specific character to you terminal.

The best way to do this is to use the OS-9 command "DISPLAY". The DISPLAY command is rather unique. It provides a way to send characters to the "Standard Output" (that's the Terminal). You provide it with hexadecimal numbers representing ASCII characters, and it sends them to the terminal. Entering the command:

```
DISPLAY C
```

sends the Hex Code '0C' (the standard ASCII "Form Feed" Code) to the "Standard Output", which causes the screen on my Coco to clear and the cursor to go to the upper left hand corner of the screen (commonly called the "home" position). (On the ADM-3A a $1A would have cleared the screen instead of a $C.) If I enter:

```
DISPLAY 2 25 2A
```

(i.e., the Hex Codes $02 $25 $2A) my cursor goes to column 5, row 10. The "2" signals a cursor move. The X and Y coordinates are entered next with an offset of $20 added in to each. So column 5 is really $20+$05 or $25. The row is $2A for row $A or in decimal 10.

DISPLAY is a procedure. It is a 6809 object code module. At its heart is the OS-9 System Call "I$WRITE". This call sends bytes of data to a specified output. In the DISPLAY command, the output is to the standard path, which is the terminal (and since it IS output to the "Standard Output", it can be "redirected" anywhere). The bytes are from the parameter list that you specify. I$WRITE doesn't care what you give it. It sends data as it is received. In Listing 2, I used I$WRITE System Call. I'll tell you more about it later.

One of the things about OS-9 that is really great is the parallelism that runs throughout it. If you are in Basic09 and you want to use I$WRITE, you use a command called "PUT". PUT needs a path and some variable or structure to send (i.e., 'where' do you want to send 'what'). If you wanted send the $C to clear the screen you could use a procedure like "Home". In Basic09, it might look like this:

```
Procedure Home
DIM ch,stdout: BYTE
stdout := 1
ch := $C
PUT #stdout,ch
END
```

This little procedure does what we did

ASSEMBLERS

ASTRUK09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the **TSC Macro Assembler**. F, CCF - **$99.95**

Macro Assembler for **TSC** -- The FLEX **STANDARD** Assembler.
Special -- CCF **$35.00**; F **$50.00**

BSR Extended 6809 Macro Assembler from **Lloyd I/O**. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 **$99.00**

Relocating Assembler w/Linking Loader from **TSC**. -- Use with many of the C and Pascal Compilers. F,CCF **$150.00**

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - **$98.00**

TRUE CROSS ASSEMBLERS from **Computer Systems Consultants** -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 680a, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
FLEX, CCF, OS-9, UniFLEX each - **$50.00**
any 3 - **$100.00**
the complete set w/ C Source (except the 68000 Source) - **$200.00**

XASM Cross Assemblers for FLEX from **Compusense Ltd.** -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar **TSC Macro Assembler** Command Line and Soruce Code format, Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - **$150.00**

CRASM from **Lloyd I/O** -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- **$399.00**

CRASM 16.32 from **Lloyd I/O** -- Cross Assembler for the **68000**.
FLEX, CCF, OS-9 **$249.00**

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer	SS-50 Bus (all w/ A.L. Source)
CCO (32K Req'd) Obj. Only **$49.00**	F, **$99.00**
CCF, Obj. Only **$50.00**	U, **$100.00**
CCF, w/Source **$99.00**	O, **$101.00**
CCO, Obj. Only **$50.00**.	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only **$100.00**	CCO, Obj. Only **$ 59.95**
F, " " **$100.00**	O, " " **$150.00**
	U, " " **$300.00**

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. The Single-Pass Compiler supports large Symbol Names; Variable Types; Pointers; Control Structures; Stack, A-, B-, and D-Register manipulation; etc. Includes **Source-Oriented** Debugger. F, CCF - **$198.00**

WHIMSICAL from **Whimsical Developments** -- Now supports **Real Numbers.** "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit **Integers, 6-digit Real Numbers,** unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted **as called** during compilation. Normally produces 10% less code than PL/9. F and CCF - **$195.00**

C Compiler from **Windrush Micro Systems** by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - **$295.00**

C Compiler from **Introl** -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
F, CCF, and O - **$375.00** U - **$425.00**

PASCAL Compiler from **Lucidata** -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - **$190.00** F 8" - **$205.00**

PASCAL Compiler from **OmegaSoft** -- For the **PROFESSIONAL**; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relocating Asmb. and Linking Loader.
F and CCF - **$425.00** One Year Maint. - **$100.00**

K-BASIC from **LLOYD I/O** -- A "Native Code" **BASIC Compiler** which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, **streamlined,** Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler **with** Assembler - **$199.00**

CRUNCH COBOL from **Compusense Ltd.** -- Supports large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Specfal Introductory Price (while in effect) -- **$99.95**

FORTH from **Stearns Electronics** -- A CoCo **FORTH** Programming Language. **Tailored to the CoCo!** Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multipleaing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" Tool!
Color Computer **ONLY** - **$58.95**

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

GENERAL

SOFTWARE DEVELOPMENT

Basic09 XRef from Southeast Media -- This **Basic09 Cross Reference** Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program List Utility** which outputs a fast "pretty printed" listing with line numbers. Requires Basic09 or Run8.
0 & CCO obj. only -- $39.95; w/ Source - $79.95

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source. F and CCF - $25.00

INCLUDE -- include other files in a Source Text, including Binary; unlimited nesting capabilities. F and CCF - $25.00

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation. F and CCF - $25.00

DUB from Southeast Media -- A **UniFLEX** "basic" De-Compiler. Re-Create a Source Listing from UniFLEX **Compiled** basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic. U - $219.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- **TSC Extended** BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F and CCF - $50.00, U - $75.00

DISK UTILITIES

OS-9 YDisk from Southeast Media -- For **Level I** only. Use the **Extended Memory** capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
-- **Level I ONLY** -- OS-9 obj. only - $79.95; w/ Source - $149.95

O-F from Southeast Media -- Written in **BASIC09 (with** Source), includes: **REFORMAT**, a BASIC09 Program that reformats a chosen amount of an **OS-9** disk to **FLEX** Format so it can be used normally by FLEX; and **FLEX**, a BASIC09 Program that does the actual read or write function to the special **O-F Transfer Disk**; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. O - $79.95

COPYMULT from Southeast Media -- Copy **LARGE** Disks to several smaller disks. FLEX utilities allow the backup of **ANY** size disk to any **SMALLER** size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT**. No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.
Completely documented Assembly Language Source files included.
ALL 4 Programs (FLEX, 8" or 5") $99.50

COPYCAT from Lucidata -- Pascal **NOT** required. Allows reading TSC Mini-FLEX, SSB DOS60, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in **Modular Source Code** (Assembly Language) to help solve unusual problems.
F and CCF 5" - $50.00 F 8" - $65.00

FLEX DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy** a File with **CRC Errors**; **Test Disk** for errors; **Compare** two Disks; a fast **Disk Backup** Program; **Edit Disk Sectors**; **Linearize Free-Chain** on the Disk; print **Disk Identification**; and **Sort** and **Replace** the Disk Directory (in sorted order). -- PLUS -- Ten XBASIC Programs including: A **BASIC Resequencer** with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs **Compare, Merge,** or **Generate Updates** between two BASIC Programs, check **BASIC Sequence Numbers**, compare **two unsequenced** files, and 5 Programs for establishing a **Master Directory** of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A **BASIC Cross-Reference Program**, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in **TSC BASIC, XBASIC,** and **PRECOMPILER BASIC Programs.**
ALL Utilities include **Source** (either BASIC or A.L. Source Code).
F and CCF - $90.00

COMMUNICATIONS

CHOOSER Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
FLEX, CCF, OS-9, UniFLEX; with complete Source - $100.00
without Source - $50.00

XDATA from Southeast Media -- A **COMMUNICATION** Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - $299.99

GAME

RAPIER - 6809 Chess Program from **Southeast Media** -- Requires FLEX and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the **strongest** CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - $79.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $1.50)
Add 5% Surface Foreign
10% Air Foreign

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

WORD PROCESSING

SCREDITOR III from **Windrush Micro Systems** -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SS0 DOS, OS-9 - **$175.00**

STYLOGRAPH from **Great Plains Computer Co.** -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
SPECIAL CCF and CCO - **$99.95**, F or O - **$295.00**, U - **$395.00**

SPELL from **Great Plains Computer Co.** -- Fast Computer Dictionary. Complements Stylograph.
SPECIAL CCF and CCO - **$69.95**, F or O - **$125.00**, U - **$175.00**

MAIL MERGE from **Great Plains Computer Co.** -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
SPECIAL CCF and CCO - **$59.95**, F or O - **$105.00**, U - **$175.00**

JUST from **Southeast Media** -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with ANY Editor. Supplied with "Structured Source" (Windrush PL/9); easy to see the flow of the program.

F and CCF - **$49.95**

SPELL8 "Computer Dictionary" from **Southeast Media** -- **OVER 120,000 words!** Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELL8 first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELL8 also allows the use of **Small Disk Storage** systems.

F and CCF - **$129.95**

DATA BASE - ACCOUNTING

XDMS from **Westchester Applied Business Systems** -- Powerful DBMS; M.L. program will work on a single sided 5" disk, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for **Giant** Data Bases. **XDMS Level I** provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. **XDMS Level II** adds the POWERFUL "GENERATE" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. **XDMS Level III** adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System **Manual** - **$24.95**
XDMS Lvl I - F & CCF - **$129.95**
XDMS Lvl II - F & CCF - **$199.95**
XDMS Lvl III - F & CCF - **$269.95**

ACCOUNTING PACKAGES -- **Great Plains Computer Co.** and **Universal Data Research, Inc.** both have Data Base and Business Packages written in TSC XBASIC for FLEX, CoCo FLEX, and UniFLEX.
Call 800-338-8800 for more information

MISCELLANEOUS

TABULA RASA SPREADSHEET from **Computer Systems Consultants** -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires **TSC's Extended BASIC.**

F and CCF - **$100.00**, U - **$200.00**

DYNACALC from **Computer Systems Center** -- Electronic Spread Sheet for the 6809.

F and SPECIAL CCF - **$200.00**, U - **$395.00**

FULL SCREEN INVENTORY/MRP from **Computer Systems Consultants** -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires **TSC's Extended BASIC.**

F and CCF - **$100.00**, U - **$150.00**

FULL SCREEN MAILING LIST from **Computer Systems Consultants** -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires **TSC's Extended BASIC.**

F and CCF - **$100.00**, U - **$110.00**

DIET-TRAC Forecaster from **Southeast Media** -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - **$59.95**, U - **$89.95**

*FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

before. You could PACK this procedure and put it in your commands directory, /DO/CMDS. (Remember to have RUNB, the Basic09 run-time package in the directory too.) Then instead of using DISPLAY, you can enter:

HOME

to clear the screen. You could also use it from a Basic09 procedure with a line like:

RUN HOME

The terminal is not the only direction you might want to send control characters. Many printers can also interpret special characters and adjust their output accordingly. Some printers have a large amount of things they can do if you send them the right character combination. Others have only a few simple commands in their repertoire. Most printers can handle a form feed which is $C. To send a form feed to the printer you could type:

DISPLAY C >/p

This causes the standard output to be "redirected" to the Printer instead of the Terminal. If it works, your printer should move to the top of the next page. The form feed could also be sent by a C Language routine. You can write a program called ff.c that might look like:

```
#define FF "\x0C"      /* form feed char*/
   /*************************************/
   /* This program sends a form feed    */
   /*    to the printer                 */
   /*************************************/

main()
{
    char *pathway="/p;
    int path, i;
    path=open(pathway,2);
    i=write(path,FF,1);
    close(path);
}
```

Here the function "**write**" sends the form feed character FF to the path of the Printer. The third parameter (the 1) indicates the length of FF. I'm sure you've noticed already that the "**write**" command is like the system call "I$WRITE" or Basic09's "PUT".

The best way to learn what your terminal or printer can do is to give it a try. Check their manuals for the correct control character sequences (use caution; few Manuals give you the HEX Codes - most provide ASCII Codes, which you will have to convert to Hex. If it doesn't work one way, try the other). Then you can use the DISPLAY command to send them. Now if you don't feel like typing DISPLAY each time, you might want to try the program in Listing 1. It's a Basic09 program called "**PDisplay**", which is short for "Prompted Display".

I've documented PDisplay with remarks, so it should be relatively straight forward. You can run it from Basic09 or from your commands directory after it's been packed. From OS-9 you can enter:

pdisplay

and it will return a prompt of

>>

The program will accept spaces or commas as delimiters between numbers. Also the program will allow decimal or hexadecimal numbers. The hex number must be preceded by a "$" sign so that it is interpreted correctly. All of the following examples send an escape character and an ascii "E".

```
>>27, 69
>>27 69
>>$1b, $45
>>$1b $45
```

How you want to enter them is up to you. If you want to redirect them to your printer, enter:

pdisplay >/p

from OS-9 and everything will wind up going to your printer. I should mention that you can send any numerical sequence. Ascii characters are legitimate, too. What will the following sequence send?

```
>>114 101 97 100 32 66 65 83 73 67
32 79 83 45 57
```

Try **PDisplay** and find out!

The DISPLAY command is a nice utility, but can get cumbersome. I do a lot of writing. I write this column and its listings. There is also outside writing like business letters, school work, and reports. Usually I try to dump as much as is possible in one sitting. This means form feeds being sent to the printer. Now you can type:

display c >/p

but after typing the line a dozen or so times, the line starts to look longer and longer. With this in mind I decided to create a small program to form feed the printer. My first try was the C language program I showed earlier. After it was compiled it came to about 3.5 K long. I like to pre-LOAD the program to save time, and 3.5 K is a lot of memory to waste. I eventually decided to write the program in assembly language. Listing 2 is the program titled, appropriately enough, "**ff**". It is about .5 K long counting the data

area, which is all stack space. It can be assembled using the Microware Interactive Assembler. This little program lets you see another way to send control characters. It uses the I$WRITE system call. Usually I LOAD it first into memory. Then all I do is enter

ff

whenever I want to advance the paper in the printer. If you generate a lot of paper work this will be very helpful.

Another alternative (like I said, OS-9 offers a lot of ways to accomplish something, so "pick your poison") is to build a "Procedure File" like this:

build ff

and when you get build's prompt, enter

DISPLAY C

and put this somewhere where it is handy (normally in your normal working directory, or, if you want to use it from several different places, put it under /D0). Then, again, you simply enter "ff" (or maybe /D0/ff), and when OS-9 finds that it is a procedure file, it executes it, accomplishing the same thing (you can't have a Command in your commands directory called "ff", or that will get executed instead of your Procedure File -- remember, OS-9 first looks for the command in Memory, then in the Commands Directory, and finally, in your Data Directory). Actually, this is probably the best way to go, since it is easy to change if you change Printers, for example. I have shown the different Programs to demonstrate some of the different ways you can accomplish something with OS-9, as well as provide different Programming examples of "adding bells and whistles".

Try playing around with the control characters. See what your terminal and printer can do. If you have Manuals, take a look at them, they'll tell you what your printer or terminal can do. There isn't anything you can do that will harm them. The worst thing that can happen is to hang them up so that nothing works right. Then the best solution to get things back to normal is to power down and start over. If your printer is messed up turn it off and on again. That should return it to normal. The same goes for the terminal. Give it a try. See what you can do.

Listing 1

```
PROCEDURE PDisplay

DIM s,w:STRING[80]
DIM i:BYTE
DIM j:INTEGER

(* get parameter list *)
LOOP
  PRINT ">>";
  READ #0,s

  (* exit if there are no more parameters *)
EXITIF s="" THEN
  PRINT "Pdisplay Terminated"
ENDEXIT

  (* add "e" to signal end of parameters *)
  s:=s+" e"

  (* change commas to spaces *)
  w:=""
  FOR j=1 TO LEN(s)
    IF MID$(s,j,1)="," THEN
      w:=w+" "
    ELSE
      w:=w+MID$(s,j,1)
    ENDIF
  NEXT j
  s:=w

  (* send control characters from  *)
  (* parameter list to standard output *)
  WHILE s<>"e" DO
    WHILE LEFT$(s,1)=" " DO
      s=RIGHT$(s,LEN(s)-1)
    ENDWHILE
    i:=INT(VAL(s))
    PUT #1,i
    i:=SUBSTR(" ",s)
    IF i>0 THEN
      s:=RIGHT$(s,LEN(s)-i)
    ENDIF
  ENDWHILE

ENDLOOP
END
```

Listing 2

```
*THIS PROGRAM WILL SEND A FORM FEED
*TO THE PRINTER
         NAM   FF
         TTL   FORMFEED COMMAND
         IFP1
         USE   /D0/DEFS/OS9DEFS
         ENDC
         ORG   0
*NO REAL DATA IS STORED
*BUT STACK SPACE IS RESERVED
         RMB   200
         STACK    EQU   .-1
MEMSIZ   EQU   .

*THE REAL PROGRAM STARTS HERE
         MOD   PGMEND, NAME, $11, $81, START, MEMSIZ
NAME     FCS   /FF/        MODULE NAME
START    EQU   *
         LEAX  PPATH,PCR   GET PRINTER'S NAME
         LDA   #WRITE.     TO WRITE TO PRINTER
         OS9   I$OPEN      CREATE A PATH TO IT
         BCS   ERROR       OUR REGULAR ERROR TRAPPING
         LEAX  FF,PCR      GET FORMFEED CHARACTER
         LDY   #1          THIS IS IT'S LENGTH
         OS9   I$WRITE     SEND IT TO PRINTER
         BCS   ERROR
         OS9   I$CLOSE     CLOSE PRINTER PATH
         BCS   ERROR
         LDB   #0          SO WE DON'T GET AN ERROR
ERROR    OS9   F$EXIT      WE'RE DONE
PPATH    FCC   ./P.        THE PRINTER NAME
         FCB   $0D
FF       FCB   $0C         THE FORMFEED CHARACTER
         EMOD
PGMEND   EQU   *
         END
```

CoCo User Notes

Carl Mann

COCO AS A RANDOM THOUGHT GENERATOR
Or,
HALF A THOUGHT IS (SOMETIMES)
BETTER THAN NONE

First thought: A couple of readers have written me recently with all sorts of praise for Hoyt Stearns Electronics COLORFORTH (available from Southeast Media). Now, I quite frankly haven't got any time at the moment to learn FORTH (having just become a full-time-and-a-half-and-then-some Technical Writer at a burgeoning electron accelerator factory). But here's what I can do: I can support a COLORFORTH User's Group to the extent that all readers who program in COLORFORTH (or any other flavor of the language) may write to me with permission to publish their names and addresses. There seems to be enough interest out there to warrent the effort, and I'm always eager to please. How about it, friends? (Also, how about some FORTH Articles and Programs for 68' Micro -- Editor?)

Second thought: How are computer techs trained these days? Doesn't anyone teach system repair by the "black box" approach and the half-split technique anymore? The reason I ask is that I have just come from a conversation with a young night-school attendee who sincerely hopes to become a genuine computer guru for the price of his tuition and sweat. There's only one problem (well, maybe MORE than one): the teacher is; A) using the (not forgotten and still very good but obsolete, nonetheless) TRS-80 Model 1 as the lone example of a microcomputer. B) Teaching the nit-piddly fineries of the way a clock pulse is generated before bothering to explain the importance of the power supply to the usefulness of the system, and C) teaching the class in some unheard-of Russian-American dialect! I, for one, am taken aback. Sucker that I am for the stray waif, I'm all too likely to attempt to teach it from the beginning, and forwards-instead of from the middle, and backwards.

I've said it more than once: an hour with a power supply, a chip or two, the appropriate data book, a few resistors and LEDs, and a Superstrip is almost invariably worth more than a week in a lecture hall, if you really want to learn. As for computer work: you seldom (if ever) need fancy oscilloscopes, signal generators, and such if all you want to do is fix something that worked well before it broke. All you need is a good digital voltmeter, a logic detector probe, and maybe a "logic pulser" to deal with the vast bulk of gross electrical failures. (Most of those WILL be power-supply related!) Add a good grasp of Basic Electricity, a dash of Common Sense, and the willingness to be VERY, VERY PATIENT- and you'll be a competent technician in less time than it takes to go to school for obsolete knowledge. (Of course, with no diploma to point to, you gotta learn all about Being Honest while you're at it...)

If you MUST go to school, CAVEAT EMPTOR! Do NOT be intimidated by the array of glitzy hardware (or lack thereof)! Interview the instructor! Check the guy out: does he/she speak English well? Is the discourse organized? Are all "buzzwords" (that is, "insider talk") carefully explained? Are your questions answered thoroughly, and to your complete satisfaction? IS THE REFERENCE AND TEXT MATERIAL RECENTLY PUBLISHED? (Check the date on the flyleaf.) At the rate things are changing, it doesn't pay to absorb more than the minimum system-specific knowledge necessary to pass the exams. Concentrate on the thought behind the design - on the logical reasoning which is reflected in the fact that tha darn thing (whatever it is) really does work. Fill in the details ONLY as necessary. You'll live longer, and eat better, too. Enough said. My soapbox is starting to smoke.

In an entirely unrelated matter: there is some word out thar in Rumormill Land that there's a new release of STYLOGRAH due out Sometime Soon Now... The OS-9 version should be ready as this column hits the press; the FLEX version is next in line. Major improvements in the way the keyboard works, plus a completely rewritten manual are said to be in the works. Stay tuned...

And now a word from the C. W. Mann Endowment for Computer-related Research. It has recently come to our attention that the manner in which a software manual presents the critical keystroke sequences which make the software go are NOT created equal. For example: most folks who are

confused by being told in print, say, "Press 'clear'+shift'+'up arw' to return to the top of the text" will understand the very same instructions MUCH more clearly if said instructions are written as, "Press and hold down <CLEAR>, then hold down the <SHIFT> key and press <UP ARROW> to return to the top of the text." It has something to do with the visual impact of the angle brackets upon the reader's cognitive facilities. Seems the word <SHIFT>, for example, LOOKS more like a REAL key on a REAL keyboard than a mere 'shift' ever could. As for those silly "+" signs: we at the Research Facility suggest that the novice writer of software manuals should avoid 'em like the plague. (Soner or later, the older, seasoned writers who use 'em will either retire or quit in frustration after trying to read something they forgot they wrote.) Most folks (myself included) simply don't know whether to actually press the <+> key or not, you see. So much for that.

Any questions? Write me care of 68 MJ. I proofread for cheap (free, in fact), if you don't mind the red and blue pencil marks... Send a double-spaced sample of your text with a SASE. It can't hurt.

Time to go. Until next month...

Incompatible Disks

By: S. D. Lyon
19943 Arminta St.
Winnetka, Ca. 91306
(818) 341-1244

As noted by several writers, including Ron Anderson in a previous "Flex Users Notes", there are apparently two different, incompatible standards in use when it comes to formatting 5 1/4" Flex disks in other than single-sided, single-density. This became apparent to me when I tried to read disks formatted on a DS68 machine (see Nov. 84 68 Micro Journal) on a Peripheral Technology PT69 computer. In anything other than SS-SD, all I got were read errors. Here I had two computers that could save up to 1400 sectors per disk but could "talk" to each other only via a 340 sector disk, a situation I found unacceptable.

After much disassembling of the PT69 Newdisk and a lot of head-scratching, I found that the primary difference between the two formatting techniques was the use of the side select byte in the sector preamble. Data Systems (as well as TSC, FMATE-RS, and possible Gimmix) sets the side select byte as prescribed by IBM 3740 and 34 standards. Peripheral Technology (and possible SWTP) uses the side-select byte instead as a density indicator (00 = single density, 01 = double density). Further, I found that even though Data Systems and the others set the byte per IMB standards, this byte is never checked by the READ and WRITE routines. The SEEK routine (which is called by READ and WRITE) determines the disk side logically. That is, if the sector is less than a certain number it must be on side 0 -- otherwise, it is on side 1. I reasoned, then, that if the side select byte was set as a density indicator (per Peripheral Technology), the formats should be compatible for either computer.

This was almost true! It turns out that, for some reason that I haven't been able to figure out, disks formatted by Peripheral Technology's Newdisk have never been readable by either my Data Systems or FMATE-RS (TRS-80C) computers in any format. This includes the disks supplied by Peripheral Techology with the monitor source listing and Flex adaptation routines. Rather than spend a lot of time trying to understand the PT69 formatting routine and why it didn't work on my other computers, I instead adapted the TSC Newdisk (supplied with Flex) to run on the PT69. This was not as easy as it sounds (it never is). First, the PT69 controller seems to be a lot fussier about gaps; second, there is no direct jump to SEEK as required by the TSC Newdisk. The gap problem was solved by rewriting TSC's Newdisk with the same gaps as used by the PT69. SEEK is labled READ8 (at $DE7C) in the PT69 disk drivers and can be called directly or, preferably, by another entry in the jump table at the beginning of the drivers (ORG $DE1B -- JMP READ8).

Included with this letter is a modified TSC Newdisk object code with the suggested changes to make it compatible with either type of computer. Deletions to Newdisk are by {*}; additions are indicated by comments or are obvious.

Also included is a BOOT for the PT69 written in position-independent code so as to be compatible with PTMON or any other monitor. (PTMON loads the boot in a strange place compared to other monitors.)

On the subject of Flex, I have one hint that may be useful. The area between $CA00 to $CBFF is used by Flex09 only for initialization and then is locked out by writing an RTS in the calling jump table. Flex is now unable to access this area except when rebooted, and it is now available for other purposes -- such as a resident utility. At this moment, I have a terminal driver there for the Screditor I'm using to type this letter. This would also be a good place for SAVE.CMD, eliminating the requirement for SAVE.LOW.

I hope you will find this of interest.

```
LIST NEWDISK1

                        *
                        * NEWDISK
                        *
                        * DISK FORMATTING PROGRAM FOR 6809 FLEX
                        * GENERAL VERSION DESIGNED FOR WD 1771/1791
                        *                 (WORKS WITH WD 2791)
                        *
                        * COPYRIGHT (C) 1980 BY
                        * TECHNICAL SYSTEMS CONSULTANTS, INC.
                        * PO BOX 2570, W. LAFAYETTE, IN 47906
                        *
                        * TSC NEWDISK MODIFIED FOR PT-69
                        * NOTE: THE FOLLOWING IS SUPPLIED AS A TEXT FILE ONLY
                        * YOU MUST HAVE YOUR OWN SOURCE LISTING OF TSC NEWDISK
                        * AS FURNISHED BY TSC WITH GENERAL PURPOSE FLEX.
                        * LOCATE TEXT TO BE MODIFIED BY REFERENCING
                        * NEARBY LABELS.
                        *
                        * S.D.LYON 10/31/84
                        *
                        ********************************************
                        * DISK SIZE PARAMETERS  -- 5 1/4" DRIVE
                        ********************************************
0020                    MAXTRK  EQU   40        CHANGE TO SUIT DRIVE
                        * SINGLE DENSITY
0008                    SMAXS0  EQU   10        SD MAX SIDE 0 SECTORS
0014                    SMAXS1  EQU   20        SD MAX SIDE 1 SECTORS
                        * DOUBLE DENSITY:
0012                    DMAXS0  EQU   18        DD MAX SIDE 0 SECTORS
0024                    DMAXS1  EQU   36        DD MAX SIDE 1 SECTORS

                        ********************************************
                        *  THE FOLLOWING ARE FOR 5 1/4" DISK.
                        ********************************************

                        * SIZE OF SINGLE DENSITY WORK BUFFER FOR ONE TRACK
0DAC                    TKEZ    EQU   3500
                        * TRACK START VALUE
000C                    TST     EQU   12
                        * SECTOR START VALUE
001B                    SST     EQU   27
                        * SECTOR GAP VALUE
0012                    GAP     EQU   18
                                OPT   LIS

0242 8D E7              DOSEC2  BSR   SET
0244 86 FE                      LDA   #$FE      ID ADDRESS MARK
0246 A7 80                      STA   0,X+
0248 96 28                      LDA   TRACK     GET TRACK NO.
024A A7 80                      STA   0,X+
024C D6 27                      LDB   DNSITY    DOUBLE DENSITY?
024E 27 02                      BEQ   DOSEC3    SKIP IF NOT
                        *  DELETED WITH +
                        *       LDB SIDE GET SIDE INDICATOR
0250 C6 01                      LDB   #1        *** CHANGE FROM 'AND# 01' ****
0252 E7 80              DOSEC3  STB   0,X+
                                OPT   LIS

0254 8D 94              DOSEC9  BSR   SET
0256 86 FB                      LDA   #$FB      DATA ADDRESS MARK
0258 A7 80                      STA   0,X+
025A 35 06                      PULS  D         RESTORE FORWARD LINK
025C ED 01                      STD   0,X++     PUT IN SECTOR BUFFER
025E 4F                         CLRA            CLEAR SECTOR BUFFER
025F C6 FE                      LDB   #254
0261 8D 07                      BSR   SET
0263 86 F7                      LDA   #$F7      SET CRC CODE
0265 A7 80                      STA   0,X+
0267 30 8B 12                   LEAX  GAP,X     LEAVE GAP
                        *       TST DNSITY DOUBLE DENSITY?
                        *       BEQ DOSEC9 SKIP IF NOT
                        *       LEAX GAP,X DD NEEDS MORE GAP
026B 39                 DOSECA  RTS
                                OPT   LIS
```

```
                        ********************************************
                        * SECTOR MAPS
                        * THE MAPS SHOWN BELOW CONTAIN THE CORRECT
                        * INTERLEAVING FOR A 5 INCH DISK.
                        ********************************************

0440                    SSCMAP  EQU   *
0440 01 06 04 09                FCB   1,6,4,9
0444 02 07 03 0A                FCB   2,7,3,10
0448 05 08 12 0B                FCB   5,8,18,11,16
044C 0E 13 0C 11                FCB   14,19,12,17
0450 0F 14 0D                   FCB   15,20,13

0453                    DSCMAP  EQU   *
0453 01 06 07 0A                FCB   1,6,7,10,13,16
0459 02 05 08 0B                FCB   2,5,8,11,14,17
045F 03 06 09 0C                FCB   3,6,9,12,15,18
0465 16 19 1C 1F                FCB   22,25,28,31,34
046B 13 17 1A 1D                FCB   19,23,26,29,32,35
0471 14 18 1B 1E                FCB   20,24,27,30,33,36,21
                                OPT   LIS

                        ********************************************
                        * WRITE TRACK ROUTINE
                        ********************************************
                        * WESTERN DIGITAL PARAMETERS (2791)
                        * ******* ******* ********

                        * REGISTERS:
E010                    COMREG  EQU   $E010   COMMAND REGISTER
E019                    TRKREG  EQU   $E019   TRACK REGISTER
E01A                    SECREG  EQU   $E01A   SECTOR REGISTER
E01B                    DATREG  EQU   $E01B   DATA REGISTER

E014                    DRVREG  EQU   $E014   DRIVE SELECT REGISTER
                        * COMMANDS:
00F4                    WTCMD   EQU   $F4     WRITE TRACK COMMAND

0550                    WRTTRK  EQU   *
0550 96 22                      LDA   DRN       GET DRIVE NO
0552 8D 24                      TST   SIDE      SIDE 0?
0554 27 02                      BEQ   SDRIVE    JIF SO
0556 8A 40                      ORA   #$40     ELSE SET 'SIDE 1'
0558 B7 E014            SDRIVE  STA   DRVREG   TELL CONTROLLER
055B 17 010F                    LBSR  DEL20
055E 86 F4                      LDA   #WTCMD
0560 0D 27                      TST   DNSITY    SINGLE DENSITY?
0562 27 02                      BEQ   WSNGL     JIF SO
0564 8A 02                      ORA   #2        ELSE SET DBL DENS
0566 B7 E018            WSNGL   STA   COMREG
0569 8E 0800                    LDX   #WORK     POINT TO SEC DATA
056C 17 00FE                    LBSR  DEL28
056F 20 03                      BRA   WRT1
0571 B7 E01B            WRTBYT  STA   DATREG   SEND TO CONTROLLER
0574 A6 80              WRT1    LDA   ,X+       GET DATA
0576 7D E014            WRTTST  TST   DRVREG
0579 2B F6                      BMI   WRTBYT    DRQ SET?
057B 27 F9                      BEQ   WRTTST    BUSY
057D B4 E018                    LDA   COMREG    GET ANY ERRORS
0580 39                         RTS

                        *
                        * BOOTSTRAP FLEX LOADER
                        *
                        * WRITTEN IN POSITION INDEPENDENT CODE
                        * DIRECTLY FOLLOWS NEWDISK CODE.
                        * NEWDISK AND ROM BOOT WILL RELOCATE AS REQUIRED.
                        * USES MONITOR STACK
                        *

0800                    SCTBUF  EQU   0         DATA SECTOR BUFFER

0581 20 0A              BOOT    BRA   LOAD0     EVENTUAL ORG IS $C100 (MOST SYSTEMS)
0583 20 6A              XREAD   BRA   READ      ROM CAN USE TO LOAD DIR
0585 00                 SIDF0   FCB   0         DELAY FLAG FOR SIDE TWO
0586 01                 TTRK    FCB   1         FILE START TRACK
0587 01                 TSCT    FCB   1         FILE START SECTOR
0588 00                 TDNS    FCB   0         BOOT DENSITY
0589 0800               TADR    FDB   0         TRANSFER ADDRESS
058B 0000               LADR    FDB   0         LOAD ADDRESS

058D 1F 43              LOAD0   TFR   S,U       SAVE STACK POINTER
058F 1F 34              LOAD0.5 TFR   U,S       RESTORE STACK POINTER
0591 63 8C F4                   COM   TDNS,PCR  TRY OTHER DENSITY
0594 EC 8C EF                   LDD   TTRK,PCR  SETUP STARTING TRK & SCT
0597 DD 00                      STD   SCTBUF
0599 10BE 0100                  LDY   #REC BUF+256

059D 8D 35              LOAD1   BSR   GETCH     GET A CHARACTER
059F 81 02                      CMPA  #$02      DATA RECORD HEADER?
05A1 27 10                      BEQ   LOAD2     SKIP IF SO
05A3 81 16                      CMPA  #$16      XFR ADDRESS HEADER?
05A5 26 F6                      BNE   LOAD1     LOOP IF NEITHER
05A7 8D 28                      BSR   GETCH     GET TRANSFER ADDRESS
05A9 A7 8C 00                   STA   TADR,PCR
05AC 8D 24                      BSR   GETCH
05AE A7 8C 07                   STA   T DR+1,PCR
05B1 20 EA                      BRA   LOAD1     CONTINUE LOAD
05B3 8D 1F              LOAD2   BSR   GETCH     GET LOAD ADDRESS
05B5 A7 8C 03                   STA   LADR,PCR

05B8 8D 1A                      BSR   GETCH
05BA A7 8C CF                   STA   LADR+1,PCR
05BD 8D 15                      BSR   GETCH     GET BYTE COUNT
05BF 1F 8940                    TAB             PUT IN B
05C2 27 09                      BEQ   LOAD1     LOOP IF COUNT=0
05C4 AE 8C C4                   LDX   LADR,PCR  GET LOAD ADDRESS
05C7 34 14              LOAD3   PSHS  B,X
05C9 8D 09                      BSR   GETCH     GET A DATA CHARACTER
05CB 35 14                      PULS  B,X
05CD A7 80                      STA   0,X+      PUT CHARACTER
05CF 5A                         DECB            END OF DATA IN RECORD?
05D0 26 F5                      BNE   LOAD3     LOOP IF NOT
05D2 20 C9                      BRA   LOAD1     GET ANOTHER RECORD

                        * GET CHARACTER ROUTINE - READS A SECTOR IF NECESSARY

05E1 108C 0100          GETCH   CMPY  #SCTBUF+256 OUT OF DATA?
05E5 26 0F                      BNE   GETCH4    GO READ CHARACTER IF NOT
05E7 8E 0000            GETCH2  LDX   #SCTBUF   POINT TO BUFFER
05EA EC 01                      LDD   0,X       GET FORWARD LINK
05EC 27 08                      BEQ   GO        IF ZERO, FILE IS LOADED
```

```
05EE 8D  0C          BSR   READ     READ NEXT SECTOR
05F0 26  AA          BNE   LOAD3    START OVER IF ERROR
05F2 10DE 0004       LDY   @BCTBUF+4 POINT PAST LINK
05F6 A6  A0    GETCHR LDA   0,Y+    ELSE, GET A CHARACTER
05F8 39              RTS

05F9 6E  9C 9A       JMP   [TFADR,PCR] JUMP TO TRANSFER ADDRESS

                     * READ A SECTOR FROM DISK
05FC 8D  23    READ  BSR   XSEEK    SEEK TRACK
05FE A6  8C 0        LDA   TDNS,PCR
0601 84  02          ANDA  #2       MASK DENS BIT
0603 8A  80          ORA   #$80     READ CMD, 6 MS STEPPING RATE
0605 B7  E010        STA   COMREG   SET COMMAND REG
0608 8D  7E          BSR   DEL28
060A BE  0880        LDX   #SCTBUF  POINT TO BUFFER
060D 20  05          BRA   READ3
060F B6  E010  READ2 LDA   DATREG   GET DATA
0612 A7  80          STA   ,X+
0614 7D  E014  READ3 TST   DRVREG
0617 2B  F6          BMI   READ2    DRQ SET?
0619 27  F9          BEQ   READ3    BUSY?
061B F6  E010        LDB   COMREG   GET STATUS
061E C5  1C          BITB  #$1C     MASK ERRORS
0620 39              RTS

0621 F7  E01A  XSEEK STB   SECREG   SET SECTOR
0624 8D  54          BSR   DEL28
0626 7D  FF68        TST   TDNS,PCR SINGLE DENS?
0629 26  06          BNE   XSK1     JIF NOT
062B 81  0A          CMPA  #10      SEC 1-10 ON SIDE 0 (SD)
062D 23  0A          BLS   XSK3     SIDE 0 OK
062F 20  04          BRA   XSK2     ELSE SIDE 1
0631 81  12    XSK1  CMPA  #18      SEC 1-18 ON SIDE 0 (DD)
0633 23  04          BLS   XSK3     TO SIDE 0
0635 C6  80    XSK2  LDB   #$80     SET FOR SIDE ONE
0637 58              ASLB           SET CARRY, B = $40
0638 21              FCB   $21      SKIP 2
0639 3F        XSK3  CLRB           SET FOR SIDE 0
```

```
063B F7  E014        STB   DRVREG   SET CONTROLLER
063E 24  0A          BCC   XSK4     JIF SIDE 0
0640 8D FF4E         TST   SIDF0,PCR SAME AS LAST TIME ?
0644 2B  0C          BMI   XSK5
0646 8D  23          BSR   SDELAY   ELSE DO SIDE DELAY
0648 20  08          BRA   XSK5
064A 6D  8D FF44 XSK4 TST   SIDF0,PCR SIDE 0 LAST TIME ?
064E 27  02          BEQ   XSK5
0650 8D  19          BSR   SDELAY   WAS SIDE 1, DO DELAY
0652 B1  E019  XSK5  CMPA  TRKREG   ON THIS TRACK?
0655 27  23          BEQ   DEL28    THEN DONE
0657 B7  E018        STA   DATREG   ELSE SET TARGET TRACK
065A 8D  1E          BSR   DEL28    WAIT FOR SET UP
065C C6  18          LDB   #$18     SEEK CMD
065E F7  E010        STB   COMREG   START SEEK
0661 8D  0C          BSR   HDELAY   LET HEAD SETTLE

0663 F6  E010  XWAIT LDB   COMREG   GET STATUS
0666 C5  01          BITB  #1       MASK BUSY
0668 26  F9          BNE   XWAIT    LOOP IF BUSY
066A 39              RTS

066B 63  8D FF23 SDELAY COM  SIDF0,PCR
066F 34  10      HDELAY PSHS  X
0671 8E  09C4        LDX   #2500    DELAY 20 MSEC
0674 30  1F      SDEL1 LEAX  -1,X
0676 26  FC          BNE   SDEL1
0678 35  90          PULS  X,PC

067A 17  0008  DEL28 LBSR  DEL14    ALLOW CONTROLLER TIME
067D 17  0000  DEL14 LBSR  DELA3    TO SIDEST THIS
0680 39        DELA3 RTS

                     END   NEWDISK

0 ERROR(S) DETECTED
```

PL/9 HEX Load

CLELL A. DILDY, JR.
2401 W 27th St.
Panama City, FL 32405

When using cross compilers, the object code usually ends up in some sort of formatted HEX. It not possible to directly load this hex code into memory for burning EPROM'S and such. A possible solution is to use a hex loading program that translates the hex code into binary and then loads this to memory. The enclosed utility program does this.

Having read Ron Anderson's Flex User's Notes, I decided to see if I could write the program in PL-9. Surprise, surprise, it did fit in the Flex utility space and I now have another capability at my disposal. Of course I had to trim down the library routines to make them fit in the allocated memory.

Being a HLL program, it is almost self documenting compared to an assembly language program. So if I am a little skimpy on the explanation of the program please excuse me.

Since PL-9 is a one pass compiler the main program must be the last procedure (as in Pascal). The main program calls two procedures get_files and convert_files. The purpose of get_files is to load the drive number, the file name and the extension into the file control block. This is done in a straight-forward manner, the information is read from the Flex command line. The drive defaults to the working drive and the extension defaults to .TXT.

Convert_file opens the file for read and then looks for an S. (Motorola Format). The program is then looking for a number to tell it what kind of record is present. Only S1 and S2 are supported at this time as this is all I need. If the digit is a 1 S1 is called, if a 2 S2 is called.

The procedures S1 and S2 are identical except for the number of bytes allowed for the address. S1 uses two bytes and S2 uses three. S1 and S2 set the check_sum to zero and then get the record length. This is decremented by 3 for S1 and by 4 for S2. The count for the address and the check sum must be removed. Next the address is loaded into the ex variable. The most sigificant address byte of S2 is ignored since I only have 64K in my system. Now both routines call read_record.

Read_record reads the record and places the binary numbers in memory until the record length is decremented to zero. The procedure get_hex_byte does most of the dirty work. It calls get_hex_nibble twice and combines the two hex digits into a byte. It also takes care of the check sum.
The check sum is checked in the next to last three lines of read_record. Since the last byte of the record is the ones complement of the check sum then the check sum of the whole record should be -1. This is set to zero by adding a +1 to the sum and then checked. If a check sum error is made then the check sum error message is printed, but the program

continues on. An improvement can be made at this point, you may want to know what line had the error. This is a little early to tell if this will really be necessary.

I hope this program will be of some interest to you and your readers.

```
/*****************************************
*                                       *
*          A HEX LOAD PROGRAM           *
*                By                     *
*          Cleit  Diddy                 *
*                                       *
*            12-27-84                   *
*                                       *
*****************************************/

ORIGIN = $C100;

INCLUDE 1.IOS
INCLUDE 1.FLEX

AT $CB40: BYTE FCB,ERROR(319);
AT $0000: BYTE MEMORY;

PROCEDURE NOTHING;
ENDPROC;

PROCEDURE GET_FILE_CHAR: BYTE CHAR;
   CHAR = READ(.FCB);
   IF ERROR .AND ERROR <> 8 THEN REPORT_ERROR(.FCB);
ENDPROC BYTE CHAR;

PROCEDURE GET_HEX_NIBBLE:BYTE INCHAR;
   INCHAR = GET_FILE_CHAR;
   IF INCHAR >= '0 .AND INCHAR <= '9
      THEN INCHAR = INCHAR - '0;
   IF INCHAR >= 'A .AND INCHAR <= 'F
      THEN INCHAR = INCHAR - '7;
ENDPROC BYTE INCHAR;

PROCEDURE GET_HEX_BYTE(BYTE .CHECK_SUM):BYTE INCHAR;
      INCHAR = SHIFT(GET_HEX_NIBBLE,4);
      INCHAR = INCHAR OR GET_HEX_NIBBLE;
      CHECK_SUM = CHECK_SUM + INCHAR;
ENDPROC BYTE INCHAR;

PROCEDURE GET_HEX_ADDRESS(BYTE .CHECK_SUM):INTEGER INCHAR;
      INCHAR = SWAP(INTEGER(GET_HEX_BYTE(.CHECK_SUM)));
      INCHAR = INCHAR OR INTEGER(GET_HEX_BYTE(.CHECK_SUM));
ENDPROC INCHAR;

PROCEDURE READ_RECORD(INTEGER EX: BYTE RECORD_LENGTH, CHECK_SUM);
   WHILE RECORD_LENGTH > 0
      BEGIN
         MEMORY(EX) = GET_HEX_BYTE(.CHECK_SUM);
         RECORD_LENGTH = RECORD_LENGTH - 1;
         EX = EX + 1;
      END;
   GET_HEX_BYTE(.CHECK_SUM);
   CHECK_SUM = CHECK_SUM + 1;
   IF CHECK_SUM THEN PRINT("\NCHECK SUM ERROR !!!\N\N");
ENDPROC;

PROCEDURE S1: BYTE CHECK_SUM, RECORD_LENGTH: INTEGER EX;
   CHECK_SUM = 0;
   RECORD_LENGTH = GET_HEX_BYTE(.CHECK_SUM) - 3;
   EX = GET_HEX_ADDRESS(.CHECK_SUM);
   READ_RECORD(EX,RECORD_LENGTH,CHECK_SUM);
ENDPROC;

PROCEDURE S2: BYTE CHECK_SUM, RECORD_LENGTH: INTEGER EX;
   CHECK_SUM = 0;
   RECORD_LENGTH = GET_HEX_BYTE(.CHECK_SUM) - 4;
   GET_HEX_BYTE(.CHECK_SUM);        /* FIRST BYTE HAS TO BE ZERO */
   EX = GET_HEX_ADDRESS(.CHECK_SUM);
   READ_RECORD(EX,RECORD_LENGTH,CHECK_SUM);
ENDPROC;

PROCEDURE CONVERT_FILE: BYTE CHAR;
   OPEN_FOR_READ(.FCB);
   IF ERROR THEN
      BEGIN
         REPORT_ERROR(.FCB);
         CLOSE_FILE(.FCB);
         RETURN;
      END;
   REPEAT
      REPEAT UNTIL GET_FILE_CHAR = 'S .OR ERROR;
      IF ERROR = 0 THEN
         BEGIN
            CHAR = GET_FILE_CHAR;
            IF ERROR = 0 THEN
               IF CHAR
                  CASE NUL THEN NOTHING;
                  CASE '0  THEN NOTHING;
                  CASE '1  THEN S1;
                  CASE '2  THEN S2;
                  CASE '3  THEN NOTHING;
                  CASE '5  THEN NOTHING;
                  CASE '7  THEN NOTHING;
                  CASE '8  THEN NOTHING;
                  CASE '9  THEN NOTHING;
               ELSE ERROR = TRUE;
         END;
   UNTIL ERROR .OR GETKEY = ABT;
   CLOSE_FILE(.FCB);
ENDPROC;

PROCEDURE GET_FILES;
   GET_FILENAME(.FCB);
   GET_EXTENSION(.FCB,TXT);   /* DEFAULT EXTENSION IS .TXT */
   IF ERROR THEN
      BEGIN
         REPORT_ERROR(.FCB);

         RETURN;
      END;
ENDPROC;

PROCEDURE MAIN;
   GET_FILES;
   CONVERT_FILE;

/* IOSUBS.LIB TRIMMED DOWN TO SAVE CODE */

CONSTANT CR=$0D,LF=$0A,SP=$20,BS=$08,NEXT=$0E,CLR=$1A,EOT=$04,
         NUL=$00,ABT=$03,CAN=$18,BEL=$07,ESC=$1B,TAB=$09,
         TRUE=-1,FALSE=0,TXT=1,INBUFF=$CD1B,DOCMND=$CD48,
         XON=$11,XOFF=$13;

AT $E004: BYTE TERM(2);

PROCEDURE PUTCHAR(BYTE CHAR);
   REPEAT UNTIL TERM AND 2;
   TERM(1) = CHAR;
ENDPROC;

PROCEDURE CRLF;
   PUTCHAR(CR);
   PUTCHAR(LF);
ENDPROC;

PROCEDURE PRINT(BYTE .STRING): BYTE CHAR;
   WHILE STRING
      BEGIN
         IF STRING = '\ THEN
            BEGIN
               .STRING = .STRING+1;
               IF STRING >= 'a .AND STRING <= 'z
                  THEN CHAR = STRING - $20;  /* CONVERT TO UPPER CASE */
                  ELSE CHAR = STRING;
               IF CHAR
                  CASE 'N THEN CRLF;
                  CASE 'B THEN PUTCHAR(BEL);
                  CASE 'R THEN PUTCHAR(CR);
                  CASE 'E THEN PUTCHAR(ESC);
                  CASE '0 THEN PUTCHAR(NUL);
               ELSE BEGIN
                    PUTCHAR('\);
                    PUTCHAR(CHAR);
                    END;
            END;
         ELSE PUTCHAR(STRING);
         .STRING = .STRING+1;
      END;
ENDPROC;

PROCEDURE GETCHAR;
   REPEAT UNTIL TERM AND 1;
ENDPROC BYTE TERM(1) AND $7F;

PROCEDURE GETKEY;
   IF TERM AND 1 THEN RETURN TERM(1) AND $7F;
ENDPROC BYTE 0;

      /* FLEX.LIB TRIMMED DOWN TO SAVE CODE */

      ASMPROC GET_FILENAME(INTEGER);
         GEN $AE,$62;          /*        LDX   2,S       */
         GEN $BD,$CD,$2D;      /*        JSR   $CD2D     */
         GEN $25,$02;          /*        BCS   *+4       */
         GEN $6F,$01;          /*        CLR   1,X       */
         GEN $39;              /*        RTS             */

      ASMPROC GET_EXTENSION(INTEGER,BYTE);
         GEN $AE,$63;          /*        LDX   3,S       */
         GEN $A6,$62;          /*        LDA   2,S       */
         GEN $BD,$CD,$33;      /*        JSR   $CD33     */
         GEN $39;              /*        RTS             */

      ASMPROC REPORT_ERROR(INTEGER);
         GEN $AE,$62;          /*        LDX   2,S       */
         GEN $7E,$CD,$3F;      /*        JMP   $CD3F     */

      ASMPROC OPEN_FOR_READ(INTEGER);
         GEN $AE,$62;          /*        LDX   2,S       */
         GEN $B6,$01;          /*        LDA   $1        */
         GEN $A7,$84;          /*        STA   ,X        */
         GEN $7E,$D4,$61;      /*        JMP   FMS       */

      ASMPROC READ(INTEGER): BYTE;
         GEN $34,$10;          /*        PSHS  X         */
         GEN $AE,$64;          /*        LDX   4,S       */
         GEN $BD,$D4,$06;      /*        JSR   FMS       */
         GEN $1F,$89;          /*        TFR   A,B       */
         GEN $35,$90;          /*        PULS  X,PC      */

      ASMPROC CLOSE_FILE(INTEGER);
         GEN $AE,$62;          /*        LDX   2,S       */
         GEN $B6,$04;          /*        LDA   $4        */
         GEN $A7,$84;          /*        STA   ,X        */
         GEN $7E,$D4,$0A;      /*        JMP   FMS       */
```

Upgrading My /09

Hidemasa Kitazume
2-3-8 Mukogaoka 1204
Bunkyo-ku, Tokyo 113
JAPAN

I would like to report my further experience with my /09 computer from SWTPc. When I wrote my struggle with /09 in Feb. 1982 '68' Micro. my system was

1) /09 from SWTPc with 64 K RAM running at 2 Mhz.
2) Calcomp 143 8 inch disk drives with DMFA2 disk controller board.
3) H-9 Heath kit terminal, modified to 24 lines.
4) Microline 80 printer from Oki Data.
5) IBM selectric printer.
6) FLEX09 DOS form SWTPc.
7) NEWDISK.CMD of FLEX worked only at 1 Mhz clock.

NEWDISK command at 2 Mhz

I happened to buy another 6809 CPU board from SWTPc. I inserted the board to my /09 system to check the board and I found that NEWDISK.CMD can run with that board at 2 Mhz clock. The board was further up graded and called as MP-09B. The difference I could be aware of was pull up resistors (4.7 K) at A8-A16 of 6809 CPU chip. If someone have the same kind of problem I suggest to get contact with SWTPc or try 8 pull up resisters.

VIDEO BOARD

I had been using H9 terminal from Heath Kit for years. When I purchased as a kit it was the cheapest and probably the best available. Yet, I was not satisfied with 80X12 character display so I modified to 80X24 display. To do that I had to sacrifice transfer speed at 600 baud. Now I was frustrated with the speed. I needed better one especially when I was using RMS data base management which uses screen cursor control a lot. Memory mapped video boards appeared to be attractive but I could not afford to let them eat up some of the memory space of my system. I realized that F&D associates released new video board which sits at one of the I/O slots of SS-30 bus and occupies only 4 addresses. The board can be made very economically and yet powerful and flexible. The video board driver was written in machine language for 6800 but easy to be modified for 6809. There are a few points that I noticed. The 7400 at IC11 must be 7400 and not 74LS00 for stable oscillation. Some extra timing loop is necessary in the software for scroll of the line when software is run at 2 Mhz. I finally incorporated the board to my /09 system in following manner.

1)the board was inserted at 3 I/O slot at address E030.
2)the board driver was configured as 80X28 characters display.
3)the board driver was burned into EPROM and placed at E800 of the CPU board, the initial part of the program is shown at Appendix 1.
4)SBUG-E was modified so that OUTCH jump to the board driver as shown at Appendix 2.
5)one scratch pad RAM was placed from F400-F7FF at CPU board. The way I did is shown in Fig. 1.

I am very satisfied with the board considering its performance and cost. I could communicate with the computer much faster.

EXTENDED ADDRESSING IN /09 WITH FLEX09 FROM SWTPc.

Since after I saw the advertisement of VDISK virtual disk software in '68' micro journal and I felt it needed, I began worrying if my own FLEX DOS run at extended addressing. I experimented to modify the CPU, DMFA2 disk controller and 64K RAM cards for extended addressing. After power up the computer, a message "SBUG1.8-16 K" was displaied. I tried to boot the FLEX and I was surprised seeing that FLEX could be booted and prompted as "FLEX-Version2.8:2-64K RAM". Now I was quite sure that I could expand RAM and use VDISK.

64 K RAM BOARD FROM DIGITAL RESEARCH

The first thing I had to do was to expand RAM. I considered that 64 K RAM board form Digital Research is the most economical and reliable (using static RAM). I found that I was right. So far I bought three bare boards and fully filled with 6116 type RAMs which was getting cheaper. Finally I made 256 K RAM computer with extended addressing.

VDISK FORM SOUTHEASTERN MICRO SYSTEMS

Then I was ready to get the virtual disk software and I found that it was really useful. My BASIC programs such as statistical analysis and information management have a lot of disk access for temporally data storage and the execution time became much shorter by using virtual disk memory. I am fully satisfied with the software and I believe that every one who wants to manipulate a lot of data should consider to have it.

JOB CONTROL PROGRAM (JCP)

After up grading from SWTPc 6800 with FLEX and SWTPc disk BASIC to SWTPc 6809 /09 system, I noticed that sequential data handling of both BASIC are different. To transfer the data that I stored using 6800 to current system, I needed frequent modification of the data using text editor. I thought that JCP could help me. So I got the program. Sure it helped me a lot. My computer kept working for hours without any direct order from me. Computer is to be like this from original thought (Automaton). If we combine with other programs such as FLEX Command, data base manager, editor, processor, BASIC and JCP itself, we really able to make our computer do a lot without any direct communication with it for a while. I noticed that the price was reduced recently and I believe that it is really worth to pay for.

THE LATEST FLEX

My current FLEX09 is Ver.2.8:3 and I do not know if it is the latest version at this moment. When I got it by purchasing FLEX DISK only at $10.00 form SWTPc some time ago, I noticed a few addition of the commands. One of the useful new commands is NF.CMD which adds the necessary printer paper feed to complete fixed pages whenever the printer is used. I appreciate if somebody review such a new development periodically for users. I failed to notice about NF.CMD in TSC FLEX news letters or any pages in '68' micro journal. I hope I missed it.

PRINTER BUFFER

In spite of the memory expansion and

virtual memory disk and speedy CRT display, I still yet have to wait impatiently quite often. That is slow printer (only 80 characters per min.). I decided to make a printer buffer. Using my old 32 K RAM board, CPU board from Data System '68' and I/O board from F&D associates, I made 32 K + 2 K RAM printer buffer. Data storage RAM is from 0000 to 7FFF and PIA is placed at F070 and scratch pad memory was placed at D000 to use SBUG-E for system check up. I will send the software if someone request.

FLEX09 FROM TSC

I never be able to boot the FLEX from TSC in spite of their effort to modify for my slow Calcomp disk drives. I was very unhappy and Mr.Daniel E. Vanda, Vice President of TSC seemed to be angry to my impatience. I finally gave up. What the waste of time, money and emotional activity.

MY FINAL SYSTEM

Let me review my final system at this moment.

1)6809 /09 SWTPc computer with 256 K static RAM at 2 Mhz. clock.
2)AGA-1 video board at E030 with 80X28 display.
3)Heath Zenith video monitor.
4)32 K RAM printer buffer.
5)Microline 80 printer.
6)Selectric typewriter printer.
7)Calcomp 8 inch disk drives with DMFA2 disk controller.
8)FLEX09 from SWTPc.
9)VDISK.CMD from Southeastern Micro.
10)JOB CONTROL PROGRAM.
11)A/BASIC compiler and TSC BASIC interpreter.
12)SPELL'N FIX from Star Kits with EDITOR and PROCESSOR form TSC.
13)RMS data base management from Washington Computer Services.

I have been enjoying using computer and it is really helping me in data base management, statistical analysis and word processing. My recent research was greatly augmented by computer use. My current project is to develop software to do data management and handling and reporting for my catheterization laboratory. I was asked to try to develop that system using my computer before our laboratory decide to pay 0.3 million dollars for new computer installation. I am quite sure that my system can do as such a expensive equipment does. I am also trying to develop another 6809 computer system by modifying old SWTPc 6800 chassis and mother board. I am planning to put 6809 CPU with 256 K dynamic RAM at 1 Mhz. clock with AGA-1 video board and GIMIX SS-30 disk controller with 8 inch FDD-8 Siemens Desk Drives. (I do not want to install my computer in the lab. so I have to make another one.)

The best thing that SWTPc have done is to make /09 system to have extended addressing. We can expand the system a lot in spite of 16 bit address lines from 6809 CPU chip. They have been using new chips such as 4 K RAM with 6800 system, 2716 EPROM when they released EPROM programmer (at that time one 2716 cost about $40) and then 64 K RAM when they introduced 6809 system. I feel that they are always looking for the future.

I recently purchased PB-4 printer buffer board from Acorn computer. I was very surprised with their sincere since they sent me new version of software in ROM when they found a bug. We need such a company who cares for us. Similar appreciation goes to Data System '68', F&D associates, Star Kits, Digital Research and SWTPc. To be honest with you I had bad experience with TSC and I do not want to express my feeling in this article. Although another new journal for 68XX was published I was somewhat disappointed. Final thanks should go to '68' micro journal. It is not fancy with high quality print but yet has a lot of useful information in more or less informal environment, something that we feel at home, yes that is "hobby world", relaxed from pressure we have during our work etc,etc.

ADDITIONAL NOTE

Above note was written 2 years ago before I came back to Japan. Since then I learned a lot about my system and added two 8 inch disk drives with 3 msec. seek speed, modified DMFA1 disk controller board for extended addressing and added a digitizer. I made a program for catheterization record management system and using daily. I found a few low cost 6809 pc boards in Japan and very useful. I will report about to '68' micro journal.

Appendix 1

```
*
*RAM SCRATCH PAD AREA
*
 ORG $F400
LINE RMB 1
CHAR RMB 1
XHLD RMB 2
SAVEX RMB 2
SAVEX1 RMB 1
SAVEX2 RMB 1
POSSTG RMB 2
XACC RMB 2
ESCFLG RMB 1 FLAG FOR ESCAPE SEQUENCES
SAVEU RMB 2 USE SP TMEPORALY SAVE
CK RMB 2
INCHE EQU $FDCC
*
* EQUATES
*
VIDADD EQU $E020 BEGINNING ADD. OF CONTROLLER

LINLEN EQU 80 LENGTH OF LINE
ESCHAR EQU $1B ESCAPE CHARACTER
*
*ERASE TO END OF LINE EQUALS CNTL/U
*ERASE TO END OF SCREEN EQUALS CNTL/V
*
*
*
* LIST OF ENTRY
* START0: DIRECT ENTRY FOR INITIALIZATION
* WRIT1 : USUAL ENTRY FROM SBUGE
*         INITALIZE AT FIRST ENTRY
* WRIT3 : ENTRY WITHOUT INITSLIZATION AT ALL
* INCHEE: INCHE THEN PRINTOUT CAHRACTER
*
 ORG $E800
ADD1 FDB START0
ADD2 FDB WRIT1
ADD3 FDB WRIT2
ADD4 FDB WRIT3
ADD5 FDB INCHEE
START0 BSR USAVE
 BSR START
 BRA ULOAD
```

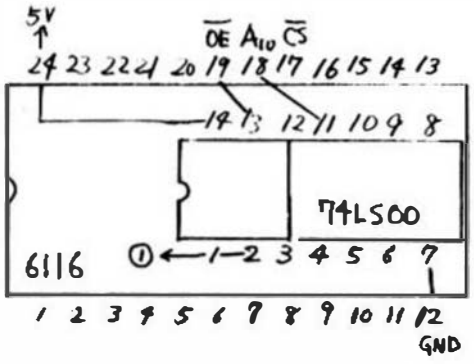
```
START LBSR VIDEO
 CLR ESCFLG
 LDAA 0M HOME UP CURSOR
 LBSR VCHAR
 LDAA 0S CLEAR SCREAN
 LBSR VCHAR
 LDAA $1B GET NEXT PAGE
 LBSR VCHAR
 LDAA 'V
 LBSR VCHAR
 LDAA 0S CLEAN UP NEXT PAGE
 LBSR VCHAR
 LDAA 0M GET BEGINING OF MEMORRY
 LBSR VCHAR
 RTS
WRIT3 BSR USAVE
WRIT4 ANDA $7F
 BRA WRIT5
WRIT2 BSR USAVE
WRIT5 LBSR VCHAR
 BSR ULOAD
 RTS
INCHEE JSR INCHE
 BSR WRIT1
 RTS
WRIT1 BSR USAVE
 LDB $55
 CMPB CK
 BNE START1
 COMB
 CMPB CK+1
 BEQ WRIT4

START1 LDB $55
 STB CK
 COMB
 STB CK+1
 BSR START
 BRA WRIT4
* USER STACK POINTER HANDLING
USAVE STU SAVEU
 LDU $F7FF
 PSHU A,B,X,Y
 RTS
ULOAD PULU A,B,X,Y
 LDU SAVEU
 RTS
*
*ACTUAL DRIVER ROUTINES START HERE
*
```

Appendix 2

For using AGA-1 video controller it is necessary to patch SBUG-E monitor ROM.

SBUG-E Ver. 1.8

FDF1 6E 9F E802 OUTCH JMP <ADD2



① point D of S1 of MP-09B

Fig. 1.

Appendix 3.

```
                    * PRINT SPOOLER PROGRAM
                    * FOR 68HK SYSTEM
                    * DATE 7/13/82
                    *
                    * EQUATES
                    *
              0001  PPCR    EQU   1
              0000  PDDR    EQU   0
              0000  PDR     EQU   0
              0003  IICR    EQU   3
              0002  IDDR    EQU   2
              0002  IDR     EQU   2
              7FFF  SMAX    EQU   $7FFF
              DFB0  STACK   EQU   $DFB0
                    *
                    * RMBS
                    *
D800                        ORD   $D800
D800                POUT    RMB   2
D802                PINP    RMB   2
D804                MMAX    RMB   2
D806                PFLAG   RMB   1
D807                IFLAG   RMB   1
                    *
                    * START PROGRAM
                    *
                    * INITIALIZATION
                    *
F800                        ORG   $FB00
F800 20   03        START   BRA   INIT
F802 01             VER     FCB   1
F803 E070           PIA     FDB   $E070
F805 108E D800      INIT    LDY   #POUT
F809 8E   0000              LDX   #$0000
F80C AF   A1                STX   0,Y++
F80E 30   01                LEAX  1,X
F810 AF   A1                STX   0,Y++
F812 8E   7FFF              LDX   #SMAX
F815 AF   A1                STX   0,Y++
F817 86   FF                LDA   #$FF
F819 A7   A0                STA   0,Y+
F81B A7   A0                STA   0,Y+
F81D 8D   4D                BSR   POPEN
F81F 17   0080              LBSR  IOPEN
                    *
                    * MAIN LOOP
                    *
F822 17   0093      INPUT1  LBSR  ICHECK
F825 7D   D807              TST   IFLAG
F828 2B   1D                BMI   PRINT1
F82A 10BE D802              LDY   PINP
F82E 10BC D800              CMPY  POUT
F832 27   13                BEQ   PRINT1
F834 17   0092              LBSR  INPUT
F837 A7   A0                STA   0,Y+

F839 10BC D804              CMPY  MMAX
F83D 23   04                BLS   STORY
F83F 108E 0000              LDY   #$0000
F843 10BF D802      STORY   STY   PINP
F847 10BE D800      PRINT1  LDY   POUT
F84B 31   21                LEAY  1,Y
F84D 108C D804              CMPY  MMAX
F851 23   04                BLS   PRINT2
F853 10BE 0000              LDY   #$0000
F857 10BC D802      PRINT2  CMPY  PINP
F858 27   C5                BEQ   INPUT1
F85D 8D   30                BSR   PCHECK
F85F 7D   D806              TST   PFLAG
F862 27   BE                BEQ   INPUT1
F864 8D   16                BSR   PRINT
F866 108F D800              STY   POUT
F86A 20   B6                BRA   INPUT1
                    *
                    *
                    * PRINTER INITIALIZATION
                    *
```

```
F86C AE   8C 94     PDPEN   LDX     PIA.PCR
F86F C6   3A                LDB     #$3A
F871 E7   01                STB     PPCR.X
F873 C6   FF                LDB     #$FF
F875 E7   84                STB     PDDR.X
F877 C6   3E                LDB     #$3E
F879 E7   01                STB     PPCR.X
F87B 39                     RTS
                    *
                    * PRINT RDUTINE
                    *
F87C 7F   D806      PRINT   CLR     PFLAG
F87F AE   8C 81             LDX     PIA.PCR
F882 A6   A4                LDA     0.Y
F884 A7   84                STA     PDR.X
F886 C6   36                LDB     #$36
F888 E7   01                STB     PPCR.X
F88A C6   3E                LDB     #$3E
F88C E7   01                STB     PPCR.X
F88E 39                     RTS
                    *
                    * PRINTER CHECK ROUTINE
                    *
F88F 7D   D806      PCHECK  TST     PFLAG
F892 2B   0D                BMI     PCEXIT
F894 AE   8D FF6B           LDX     PIA.PCR
F898 6D   01                TST     PPCR.X
F89A 2A   05                BPL     PCEXIT
F89C 6D   84                TST     PDR.X
F89E 73   D806              COM     PFLAG
F8A1 39             PCEXIT  RTS
                    *
                    * INPUT INTIALIZATION
                    *
F8A2 AE   8D FF5D   IDPEN   LDX     PIA.PCR
F8A6 C6   3A                LDB     #$3A
F8A8 E7   03                STB     IICR.X
F8AA 5F                     CLRB
F8AB E7   02                STB     IDDR.X
F8AD C6   36                LDB     #$36
F8AF E7   03                STB     IICR.X
F8B1 E6   02                LDB     IDR.X
F8B3 C6   3E                LDB     #$3E
F8B5 E7   03                STB     IICR.X
F8B7 39                     RTS
                    *
                    * INPUT CHECK ROUTINE
                    *
F8B8 7D   D807      ICHECK  TST     IFLAG
F8BB 27   0B                BEQ     ICEXIT
F8BD AE   8D FF42           LDX     PIA.PCR
F8C1 6D   03                TST     IICR.X
F8C3 2A   03                BPL     ICEXIT
F8C5 73   D807              COM     IFLAG
F8C8 39             ICEXIT  RTS
                    *
                    * INPUT RDUTINE
                    *
F8C9 AE   8D FF36   INPUT   LDX     PIA.PCR
F8CD A6   02                LDA     IDR.X
F8CF C6   36                LDB     #$36
F8D1 E7   03                STB     IICR.X
F8D3 C6   3E                LDB     #$3E
F8D5 E7   03                STB     IICR.X
F8D7 73   D807              COM     IFLAG
F8DA 39                     RTS
F900                        ORG     $F900
F900 10CE DFB0      SETUP   LDS     #STACK
F904 16   FEFE              LBRA    INIT
FFFA                        ORG     $FFFA
FFFA F900                   FDB     $F900
FFFC F900                   FDB     $F900
FFFE F900                   FDB     $F900

0 ERRDR(S) DETECTED
SYMBOL TABLE:

ICEXIT F8CB  ICHECK F8B8  IDDR   0002  IDR    0002  IFLAG  D807
IICR   0003  INIT   F900  INPUT  F8C9  INPUT1 F8D2  IDPEN  F8A2
PAUX   D004  PCEXIT F8A1  PCHECK F88F  PDDR   0000  PDR    0000
PFLAG  D006  PIA    F800  PINP   D002  POPEN  F86C  POUT   D000
PPCR   0001  PR'NT  F87C  PRINT1 F8A7  PRINT2 F857  SETUP  F900
STACK  7FF9  STACK  DFB0  START  F800  STOAV  F8A3  VER    F002
```

Match Utility

This is a file compare utility which is specialized for text files. When a difference is found the utility attempts to rematch the files. A rematch requires that 4 consecutive lines be identical. Once the rematch is found then all the different lines are printed on the terminal. One identical line either side of the different lines is also printed.

The utility may be aborted at any time by typing control C. If a difference consists of many extra lines in one file or if the two files are totally different, then it may take several seconds or even minutes trying to rematch. For this reason the utility tells you when it is trying to rematch. Again you may abort it by typing a control C.

Invoke the utility by typing:

MATCH,filename,filename

This utility is found on Program Disk #17 as sold by 68' Micro Journal. See page 60.

```
%                    MATCH UTILITY
%                    XXXXXXXXXXXX

% WRITTEN : MAR 12TH 1985
% AUTHOR  : JOHN SPRAY
%           8 VALLEY VIEW ROAD, GLENFIELD,
%           AUCKLAND, NEW ZEALAND.
%           PH (64) 19) 4446550
% LANGUAGE : NON-TECHNICAL

^TITLE "MATCH UTILITY"
^STACK=ENDCZ80; % stack begins from ssssnd
^VERBTOM=T
BEGIN
  CHAR FILE FILE1,FILE2      % THE TWO FILES TO BE MATCHED
  DBYTE BUFSZ=$2000;
  CHAR ARRAY BUFF1(BUFSZ), BUFF2(BUFSZ); % TWO LARGE CIRCULAR BUFFERS
  DBYTE HEAD1,HEAD2,         % HEAD OF THE TEXT IN BUFFERS
        WORK1,WORK2,         % WORK PART OF BUFFERS (USUALLY ONE LINE BEYOND HEAD)
        TAIL1,TAIL2;         % TAIL OF TEXT IN BUFFERS

  DBYTE NEXT1,NEXT2;         % NEXT LINES TO BE COMPARED
  BOOLEAN FINISH,            % REACHED THE END OF THE FILES
          DIFFERENCE,        % DIFFERENCE FOUND FLAG
          CANT;              % CANT REMATCH WITHIN BUFFER

  PROCEDURE PCRLF=EXTERNAL($CD24); % CALL FLEX PCRLF SUBROUTINE

  BOOLEAN PROCEDURE STATUS=  % DETERMINE IF KEY PRESSED
  CODE($00,$CD4E,            % JSR STAT
       $2701,                % BEQ CLRB
       $C4,                  % LDB #$FF
       $5F,                  % CLRB
       $39);                 % RTS

  PROCEDURE REPEAT(CHAR CH)=
  BEGIN
    SMALLINT I;
    FOR I:=1 TO 36 DO WRITE CH;
  END;

%-------------------------------------------------------------
% Procedures READLINE1 and READLINE2 read a full line from FILE1
% or FILE2 respectively into the appropriate buffer at TAIL1 or
% TAIL2. The variable NEXT1 or NEXT2 is also updated to point to
% the end of the new lines.

  PROCEDURE READLINE1;
  BEGIN
    CHAR CH;
    DO BEGIN
      IF EOF(FILE1) THEN CH:=CHR($00) ELSE READ FROM FILE1 CH;
      BUFF1[TAIL1]:=CH;
      TAIL1:=TAIL1+$0001;
      IF TAIL1=BUFSZ THEN TAIL1:=$0000;
      IF TAIL1=HEAD1 THEN CANT:=TRUE;
    END UNTIL CH=CHR($0D) OR CANT;
    NEXT1:=TAIL1;
  END;
```

```
PROCEDURE READLINE2;
BEGIN
  CHAR CH;
  DO BEGIN
    IF EOF(FILE2) THEN CH:=CHR(&0D) ELSE READ FROM FILE2 CH;
    BUFF2[TAIL2]:=CH;
    TAIL2:=(TAIL2+1)&7FFF;
    IF TAIL2=BUFS2 THEN TAIL2:=&0000;
    IF TAIL2=HEAD2 THEN CANT:=TRUE;
  END UNTIL CH=CHR(&0D) OR CANT;
  NEXT2:=TAIL2;
END;

%*****************************************************************
% Procedure WRITELINES is used to print lines from a buffer to
% the screen.

  PROCEDURE WRITELINES(CHAR ARRAY BUFF; BYTE FROMHERE, TOHERE);
  BEGIN
    DO BEGIN
      IF BUFF[FROMHERE]=CHR(&0D) THEN PCALL ELSE WRITE BUFF[FROMHERE];
      FROMHERE:=FROMHERE+&0001;
      IF FROMHERE=BUFS2 THEN FROMHERE:=&0000;
    END UNTIL FROMHERE=TOHERE;
  END;

%*****************************************************************
% Procedure COMPARELINES compares two lines and returns whether
% or not they are the same. The beginnings of the lines to compare
% are indexed into BUFF1 and BUFF2 respectively. If a line being
% pointed to does not set aside in the buffer then it is first
% read in from the disk file. NEXT1 and NEXT2 are set to the next
% lines on return.
% If the end of one file is reached then the other is read into
% the buffer to its end and NEXT1 and NEXT2 are set to the end of
% both texts. FINISH is set to TRUE.
% If there is not enough room in the buffer to read in a line then
% NEXT1 and NEXT2 are set to the end of the text in the buffer and
% CANT is set to true.

  BOOLEAN PROCEDURE COMPARELINES(BYTE LINE1, LINE2);
  BEGIN
    % ALSO SETS WRITE & NEXT2 TO NEXT LINES
    CHAR CH;
    BYTE L1,L2; % LENGTHS OF LINES

    NEXT1:=LINE1; NEXT2:=LINE2;
    COMPARELINES:=TRUE;
    IF EOF(FILE1) THEN
    BEGIN
      DO READLINE2 UNTIL EOF(FILE2) OR CANT;
      COMPARELINES:=FALSE;
    END;
    IF EOF(FILE2) THEN
    BEGIN
      DO READLINE1 UNTIL EOF(FILE1) OR CANT;
      COMPARELINES:=FALSE;
    END;
    IF COMPARELINES THEN

     BEGIN
       IF NEXT1=TAIL1 THEN READLINE1 ELSE
       DO BEGIN % FIND END OF LINE %
         CH:=BUFF1[NEXT1]; % GET THE CHARACTER AT LINE1
         NEXT1:=(NEXT1+&0001);
         IF NEXT1=BUFS1 THEN NEXT1:=&0000;
       END UNTIL CH=CHR(&0D);
       IF NEXT2=TAIL2 THEN READLINE2 ELSE
       DO BEGIN
         CH:=BUFF2[NEXT2];
         NEXT2:=(NEXT2+&0001);
         IF NEXT2=BUFS2 THEN NEXT2:=&0000;
       END UNTIL CH=CHR(&0D);
       IF CANT THEN COMPARELINES:=FALSE
       L1:=NEXT1-LINE1;
       IF L1<0 THEN L1:=L1+BUFS1;
       L2:=NEXT2-LINE2;
       IF L2<0 THEN L2:=L2+BUFS2;
       IF L2<>L1 THEN COMPARELINES:=FALSE;
       IF COMPARELINES THEN
       DO BEGIN
         CH:=BUFF1[LINE1];
         IF BUFF2[LINE2]<>CH THEN COMPARELINES:=FALSE;
         LINE1:=(LINE1+&0001);
         IF LINE1=BUFS1 THEN LINE1:=&0000;
         LINE2:=(LINE2+&0001);
         IF LINE2=BUFS2 THEN LINE2:=&0000;
       END UNTIL CH=CHR(&0D);
     END;
     FINISH:=(EOF(FILE1) AND EOF(FILE2)) OR CANT;
     IF CANT THEN (NEXT1:=TAIL1; NEXT2:=TAIL2);
     IF STATUS THEN (PEEK CH; IF CH=^C THEN STOP);
  END;

%*****************************************************************
% Procedure REMATCH is called when a difference is found in the two
% buffers to find where they match up again. When called, WORK1 and
% WORK2 point to the two lines which were different and NEXT1 and
% NEXT2 point to the following lines respectively.
% A search is made until 4 consecutive pairs of lines are identical.
% On exit NEXT1 and NEXT2 will point to the second pair of identical
% lines.

  PROCEDURE REMATCH;
  BEGIN
    BYTE SEARCH1:=NEXT1,  % SEARCH LINE STARTS AT NEXT
         SEARCH2:=NEXT2,
         PTR,
         MATCH1,MATCH2;   % TO REMEMBER WHERE REMATCH OCCURED
    SMALLINT GAME, REQUIRED:=3; % COUNT OF LINES SAME AND NUMBER REQUIRED
    BOOLEAN REMATCHED;

    PCRLF;
    WRITE "Rematching to resume . . . (Control C to abort)";

    DO BEGIN       % LOOK AT ONE MORE LINE IN EACH FILE
                   % CONSIDER LINE AT SEARCH1 IN FILE1
      PTR:=WORK2;  % COMPARE WITH LINES FROM WORK2 TO SEARCH2 IN FILE2
      DO BEGIN
        REMATCHED:=COMPARELINES(SEARCH1,PTR);

    MATCH1:=NEXT1; MATCH2:=NEXT2; % REMEMBER WHERE REMATCH OCCURED
    IF REMATCHED THEN % CHECK REQUIRED NUMBER OF LINES
    BEGIN
      GAME:=0;
      DO BEGIN % FIND REQUIRED NUMBER OF LINES THE SAME
        IF COMPARELINES(NEXT1,NEXT2) THEN GAME:=GAME+1 ELSE REMATCHED:=FALSE;
      END UNTIL GAME=REQUIRED OR FINISH OR NOT REMATCHED;
      IF FINISH THEN % REMATCH OCCURS AT END OF FILES
        (MATCH1:=NEXT1; MATCH2:=NEXT2);
      END;
      PTR:=MATCH1;  % INCREMENT PTR TO NEXT LINE OF FILE1
      IF FINISH THEN REMATCHED:=TRUE;
    END UNTIL PTR=SEARCH1 OR REMATCHED;
    SEARCH2:=MATCH2; % INCREMENT SEARCH POINTER TO NEXT LINE OF FILE2
  END UNTIL REMATCHED;
  NEXT1:=MATCH1; NEXT2:=MATCH2;
END;

%*****************************************************************
% The main program starts here.

  PCRLF;
  OPEN FILE1 AS "";       % OPEN FILES GETTING FILENAMES FROM COMMAND LINE
  OPEN FILE2 AS "";
  DO BEGIN
    IF NOT COMPARELINES(WORK1,WORK2) THEN % ALSO FIND NEXT1 AND NEXT2
    BEGIN
      REMATCH; % FIND NEXT1 AND NEXT2 SUCH THAT TEST IS REMATCHED THERE
      PCRLF; REPEAT("="); WRITE " FILE ONE "; REPEAT("=");
      PCALL( WRITELINES(BUFF1,HEAD1,NEXT1)); % WRITE LINES FROM HEAD1 TO NEXT1
      PCRLF; REPEAT("-"); WRITE " FILE TWO "; REPEAT("-");
      PCALL( WRITELINES(BUFF2,HEAD2,NEXT2));
      IF CANT THEN WRITE "^G^JCannot re-match these files";
      WORK1:=NEXT1; WORK2:=NEXT2; % WORK FROM NEXT
      DIFFERENCE:=TRUE;
    END;
    HEAD1:=WORK1; HEAD2:=WORK2;   % RELEASE SOME OF BUFFERS
    WORK1:=NEXT1; WORK2:=NEXT2;   % GO ON TO NEXT LINE
  END UNTIL FINISH;

  IF NOT DIFFERENCE THEN WRITE "^G^JFiles are identical";
  CLOSE FILE1;
  CLOSE FILE2;
END.
```

Bit Bucket

Dear Sirs,

In the September, '84, issue of COLOR MICRO JOURNAL, you printed my article entitled EDIT/BAS concerning a string editing routine that could be used in a BASIC program to edit file records. One limitation of the routine was that it handled only upper case characters.

I have found a simple method of adding lower case handling without slowing the operation of the routine appreciably and thought some folks might like to have the information. Also I have made several other improvements so I am including the latest version as it is found in my datafile program. This listing is set up to edit fields in a two element array.

Firstly, the original. The routine uses pokes to the text screen memory to update the cursor. On the COCO the ASCII value of an upper case character, minus 64, when poked to screen memory shows on the screen as the reverse of the character itself. I used this fact to make a non-destructible cursor by simply subtracting 64 from whatever value was PEEKed from the character location, and POKEing the new value there.

Lower case character values subtracted and POKEd this way are negative values and cause an FC ERROR. And a line to recalculate values made the routine too slow. However,

I have noted that by adding 64 to lower case instead of subtracting and POKEing that value, you get the upper case. Thus, the only difference was the sign of the value.

By using the "SGN" function, we can determine the sign of the POKE value and recalculate it, if necessary, in one quick operation.

In line 80 of the published listing the work is done by the statement "POKECA,HV-64".

Since this subtraction can give a negative value which cannot be POKEd, we want to test it for negative value. We do this by USING "SGN(HV-64)". If (HV-64) is negative the result of "SGN(HV-64)" is "-1". If (HV-64) is positive the result is "1" or "+1".

We can place the original "64" inside parentheses and change its sign by multiplying it by the "-1" or the "1" we get from the "SGN" function. Here is the new statement: "POKECA,HV-(64*SGN(HV-64))".

Actually, watching the mix of upper case and the COCO version of lowercase dance back and forth between reverse and inverse on the screen can get confusing so be careful in using this change in the routine.

In the enclosed listing some variables have been renamed for greater mnemonic clarity. The opening screen is different and the re-write routine is improved. The term "OVERSTRIKE" has been changed to "TYPEOVER". Also, since the "ENTER" key was overused the working edit mode is exited by pressing the shifted "CLEAR" key.

Line 900: "HR" holds the record number within the routine. It is either carried into line 909 of the routine from calling routines or input from keyboard at line 906. "R" is the record number counter outside the routine. Line 210 refers to a "NO DATA IN MEMORY PROMPT". You will need to substitute your record number counter for "R" and your line number for "210" if you use this check.

Line 906: The input (HR$) "M" returns you to the menu. The inputs "S" and "H" return you to other calling routines. These are the only exits from this routine. If one of these is not pressed, "HR$" becomes "HR" (Hold Record) which temporarily holds the number of the record to be edited.

Line 909: "F$(Y)" is a fieldname where "Y" is its number. You will need to substitute your field name identifier for "F$". The input "HF" holds the number of the field to be edited.

Line 912: "ER$" temporarily holds the actual text portion of the record to be edited. It is take from "R$(HR,HF)". Outside this routine, records are R$(R,F). You will need to substitute your record identifier for "R$".

This is a working form of the program previously published. All that should be necessary to make it work for you is to renumber the routine and substitute your variables and line numbers as noted above. See the original atricle for information about general use.

Jim La Fone

```
899 'edit
900 HR=0:IFR=0THEN210
903 CLS:PRINT@485,"* SEARCh mENU
 sCAN *";:PRINT@1,"edit FORMAT":
PRINTST$
906 PRINT@65,"LAST R#:"HR" ENTER
 NEXT":PRINT@90,"";:INPUTHR$:IFH
R$="M"THEN100ELSEIFHR$="S"THENX=
HR:GOTO710ELSEIFSE ANDHR$="H"THE
NX=HR:GOTO854ELSEHR=VAL(HR$):IFH
R<1ORHR>R THEN906
909 PRINT:PRINT" EDIT FIELDS:":F
ORY=1TOF:PRINTY": "F$(Y),:NEXT:I
NPUT" FIELD # ";HF:IFHF<1ORHF>F
THEN903
912 ER$=R$(HR,HF):CA=1088:CP=1
915 CLS:PRINT" edit: REC"HR" FLD
"HF:PRINTST$:PRINT@64,ER$:PRINT@
448,"iNSERT dELETE tYPEOVER rEEN
TER  nO CHANGE  eNTER  sh&cl-RE
MOVE";
918 I$=INKEY$:IFI$="R"THEN921ELS
EIFI$="\"THEN966ELSEIFI$=CHR$(13
)THEN969ELSEIFI$="N"THEN912ELSEI
FI$<>"I"ANDI$<>"D"ANDI$<>"T"THEN
918
921 IFI$="R"THENPRINT@448:PRINT:
PRINT:PRINT@259,"mode R  ENTER R
EPLACEMENT":PRINT@288,"";:LINEIN
PUTER$:GOTO915
924 CLS:PRINT" mode "I$:PRINTST$
:PRINT@332,"STOP HERE ^":PRINT@4
83,"KEYS, ALL ARROWS & <CLEAR>";
:PRINT@64,ER$:L=LEN(ER$):HV=PEEK
(CA)927 A$=INKEY$:POKECA,HV-(64*
SGN(HV-64)):IFCP<2THEN933
930 IFPEEK(343)=247THENBC=PEEK(C
A-1):POKECA,HV:HV=BC:CA=CA-1:CP=
CP-1:GOTO927
933 IFCP>L THEN939
936 IFPEEK(344)=247THENAC=PEEK(C
A+1):POKECA,HV:HV=AC:CA=CA+1:CP=
CP+1:GOTO927
939 IFA$=""THEN927
942 IFA$=CHR$(12)THEN915
945 IFA$=CHR$(10)THENIFL-CP>=32T
HENPOKECA,HV:HV=PEEK(CA+32):CA=C
A+32:CP=CP+32:GOTO927
948 IFASC(A$)<32THEN927
951 IFA$="^"THENIFCP>=33THENPOKE
CA,HV:HV=PEEK(CA-32):CA=CA-32:CP
=CP-32:GOTO927ELSE927
954 IFI$="I"THENER$=LEFT$(ER$,CP
-1)+A$+RIGHT$(ER$,L-CP+1):CA=CA+
1:CP=CP+1:GOTO924
```

```
957 IFI$="T"THENIFCP<=L THENER$=
LEFT$(ER$,CP-1)+A$+RIGHT$(ER$,L-
CP):CA=CA+1:CP=CP+1:GOTO924
960 IFI$="D"THENIFCP<=L THENER$=
LEFT$(ER$,CP-1)+RIGHT$(ER$,L-CP)
:GOTO924
963 GOTO927
966 ER$="":GOTO915
969 R$(HR,HF)=ER$:GOTO903
```

68XX Ham-Net

It seems like every month I receive a letter or phone call from one of our readers who is a Amateur Radio Operator, or more well known as a 'HAM'. Seems that a few years ago there was an informal (all should be) net or get together of 68XX users who were hams. They were able to discuss the hobby and pass along a lot of fine information and just 'rag chew' in general. As time passed it appears to have faded away. Now some want to get it started up again. I am all for it and will attempt to check in often.

Therefore, starting May 8th, 1985, a Wednesday, I will be on 75 Meters, about 3.870 Mhz, around 9:00 p.m. Eastern, looking for any of you who might like to get this thing started. I will try it each Wednesday for a while to see what response we get, however, I DO NOT want to ram-rod the thing, I just want to participate. Also please drop me a line if you think you can be there and offer some suggestions. My call is W4MQN and I will call, so please listen on the frequency.

I have also had request from quite a few readers wanting to know what others were doing, in ham radio, with their 68XX computers, CW, RTTY, Slowscan, ASCII, Satellite, etc. How about dropping me a line and let me know if you have done something and are willing to share it with the rest of us. I know for one that I would be very interested. I have all modes here but none with my own 68XX computers. I did run some CoCo stuff, but it was too crude and my HAL just did a much better job. I did receive an RTTY article a while ago but it was using the old PERCOM system and not many were left using that system so it was never run. Till then 73 Don W4MQN SK.

DMW

TALBOT MICROSYSTEMS
1927 Curtis Avenue
Redondo Beach, CA 90278
(213) 376-9941

Dear Don, Larry, Tom, Bob, ...

I am happy to see the start of Macintosh coverage, but it is surprising how quickly misinformation gets into print!

In Bud Pass's "C" User Notes of April '85 he mentions the availability and some features of the Apple assembler-language develoment system. As of this date it seems still not to be officially released, but I and many others have had a prerelease copy for several months. Bud's statement that "it requires two Macintoshes (connected together by a common external bus) or a Lisa and a Macintosh" is false.

The assembler system includes an Editor, Assembler, Linker, Debuggers, and miscellaneous source files of macros and system equates. It runs on a single drive 128K Mac (or larger, of course). It does not require two Macs nor a Mac and Lisa. Apple provides a set of several debuggers which have various sizes and capabilities. The most extensive of these debuggers does use two Macs -- one debugging the code in the other. It is supposed to be very handy, but I have worked extensively on my single Mac using the medium size debuggers and can not see any reason why one needs the two-Mac debugger.
I suspect that Bud's misinformation comes from confusion with the Lisa Pascal system. For a long time that was the only way to program the Mac. That system runs only on the Lisa and it produces code which can be moved to a Mac via RS-232 (a "common external bus").

As a point of possible interest to some of the readers, I am co-author of the Macintosh MasterFORTH system now offerred by Micromotion (p. 57, April 1985). It and a thoroughly compatible version for CP/M-68K is available right now from Talbot Microsystems; a compatible 6809 version is in the works, probably under OS-9. MasterFORTH is a family of FORTH systems for which applications can be easily transported between various machines. At present, in addition to the above, it is available for the IBM PC family (including jr), other MS-DOS systems, Apple II family, and Z80 CP/M. They have identical vocabularies, sophisticated screen editors, high level debuggers, etc. (different asssemblers of course). It is a FORTH-83 system with many sophisticated features such as headerless code generation, relocatable object code overlays, and strings. There is an optional floating-point module, graphics, and a target code compiler.

I can not help but chuckle when I read some of the articles which have been appearing in 68 Micro Journal and other magazines which mention compile times with Pascal and C compilers. The majority of benchmarks published to compare languages or hardware have been execution of the computation of primes. It is rather curious that programmers would use that to judge languages which they use day in and day out -- most of their time is spent waiting for compiles and links, very little waiting for primes.

For example, Bud mentions a program of his which took 1 hr (on Coco) and 20 minutes on FLEX (SS-50 I guess). Since he did not mention details, I can not make any comparisons. However, I know of some tasks done on a 68K machine in C which take about 20 minutes to compile and link. I do a comparable compile (no link step required) with FORTH in less than 2 minutes! Anyone who has to pay programmers will appreciate the factor of 10 savings. Of course, you could keep the sluggish languge and just buy yourself a 10 times faster computer.
Oh well, as we all know, choices of computer hardware and software are not based on reasonable decisions -- witness the explosive takeover of the "16" bit 8088 IBM's and their "compatibles."

Yours,

Ray Talbot

Ray Talbot

Bud's Comments -- 3/22/85

MDS/68 is just now being released. When I wrote the article about Apple's Assembler and C Compiler, I

had just talked to an Apple Representative on the telephone. At that time, the Assembler required two Mac's or a Lisa and a Mac to debug Software for a Mac.

Editors Note

I will add that I was told the same thing several months ago (Bud's Columns are usually written several months ahead of time), i.e., that the "present" Mac Assembler required two Macs for operation. They did not indicate, at that time, that they were "working" on a single-Mac System, or that there may have been some "Beta Test" single-Mac Assemblers out, etc.; nor would you expect ANYONE to release information on something that is in the "initial development" stage.

While I am here, I will throw my $.02 into the Programming Language/Compile Time/Benchmark pot. I don't think ANY serious Programmer would say that there is no place in the Industry for a Programming Language such as FORTH; but, neither is it the ONLY Programming Language that should be available. Much of the early Macintosh Software that is available was written in FORTH; as one Mac Software Developer put it, being able to instantly try some Mac System Call, and when the System blew up, punch the button and try another one, led to his learning the System and Developing the Program many times faster than he would have been able to with any other Language. FORTH's Interactive Environment provides fast Program Development, and its speed of execution, especially on a MPU such as the 6809 or 68000, is plenty fast enough for most applications.

The "Compile Time" and "Benchmark" comments, though, must be considered within a specific context. Where Ray mentions that Bud's C Program took 1 hr. to compile on a CoCo and 20 mins. to compile on an SS-50 Computer, while his FORTH compiled in 2 min., I would like to add that MY CoCo took NO time to compile a similar Program written in BASIC; so what have I proved?? Agreed, the Edit, Try, DeBug, re-Edit, re-Try, re-DeBug, etc., cycle is much faster with FORTH than with C or Pascal, but not necessarily as fast as Interpreted BASIC; so if the only consideration is to be "compilation time", then every one should use nothing but BASIC. For example, BASIC is excellent for writing a Program that will only be used ONCE, where the total "Life of the Program" is the time to write and execute it one time; but if rapid access to a large amount of information is required, BASIC is sure not the Language to use (If it takes longer for the Computer to find a client, for example, than it takes a Secretary to pull that Folder, why have a Computer?). Obviously, there are other considerations to be accounted for in choosing a Programming Language, such as execution Program Size, Speed of Execution, etc., besides the subjective personal criteria of simply "I am more comfortable with this Language than that Language".

This is where the "Benchmarks" come into play. Once the Program is READY TO RUN, how fast will it execute with which Languages, or which Implementations of a specific Language runs faster, or has a smaller Run-time size, etc. Again, this is only ONE consideration to be used in a Language selection, but for a "run-time" evaluation, certain benchmarks are a valid indicator of what to expect (with appropriate consideration of just WHAT the benchmark is actually evaluating). Any Application has specific requirements to meet, and many things go into determining what Language, Data Structures, Algorithms, etc., best solve the specific problem.

FORTH is a flexible and powerful Programming Language, and can be used in solving the vast majority of the Applications that a Programmer may encounter. But, like C or Pascal or BASIC or what ever, it is not the ultimate solution for all Programmers. FORTH reminds me a lot of the Macintosh: both are so different from the "normal" that we have been taught all of our lives that they take some getting used too; yet both are really fundamentally closer to the "natural order of things" than most of us realize. We would be more than happy to invite Ray, or any other "FORTH Expert", to write a Monthly Column in 68' Micro Journal on the FORTH Language, as well as invite ANY of you FORTH Users to submit Articles, Programs, Problem Solutions, or what ever for Publication. Let's hear from y'all, both Pros and Cons.

-- RLN --

Editor:

I would like to announce a new BBS (in french) called L80C, which is dedicated to TRS-80 CoCo. Features include message base, electronic shopping, downloading, uploading, games and graphics. Readers may call the BBS anytime at 418-872-8347.

J'aimerais vous annoncer un nouveau BBS, son nom est L80C, il est dedie specialement au TRS-80 Couleur. Ses possibilitees sont une systeme de messagerie, marchandise a vendre, uploading, downloading, jeu et graphique. Appelez au numeros 418-872-8347.

Welly Denoncourt
1622 Rue Alfred
Ancienne-Lorette, Que
G2E 3J1

Scintillex Software

PRESS RELEASE
For immediate release.
Date: March 15, 1985
Contact: Dan Vanada
Phone: (919) 493-7205

NEW COMPANY DEVELOPS UNIFLEX™ SOFTWARE

Chapel Hill, NC -- A new company, named **Scintillex Software**, recently announced its entry into the UniFLEX software market. The company was founded by Dan Vanada, formerly vice-president of Technical Systems Consultants, Inc. It's first two 6809 UniFLEX software packages are ready to be shipped.

The first, called **Initializer II**, replaces the standard initializer program supplied with UniFLEX. The initializer is the program which controls the booting and operation of UniFLEX. Instead of being forced to follow the fixed, unalterable sequence of the standard initializer, the **Initializer II** package allows the user to define a customized booting or operating sequence. For example, a system could be set up to automatically prompt for the date and time when UniFLEX is booted; or, it could be set up to require a password in order to boot UniFLEX. It is possible for a system to boot directly into multi-user mode, bypassing single-user mode completely. **Initializer II** also provides the ability to allow remote logins and outgoing modem calls on the same terminal port.

The second program, called the **Disk Maintenance Package**, is a set of twelve programs which assist in the maintenance of UniFLEX disks. It provides several functions which cannot be performed by software supplied with the UniFLEX Operating System. For example, one program reports what file or what part of the disk to which a specified block is allocated. Another automatically adds the "." and ".." files to a damaged directory. Yet another permits physical disk sectors, or blocks, to be read, modified, and rewritten. It performs the same functions for file descriptor nodes (fdns). The manual suggests several procedures and ideas to keep UniFLEX disk systems running smoothly.

Each program retails for $95.00. For further information, contact **Scintillex Software** at 300 Eastowne Drive, Suite 109, Chapel Hill, North Carolina 27514; (919) 493-7205.

University of Transkei
Chemistry Department
Private Bag X5092
UMTATA
Republic of Transkei
Southern Africa

Dear Don,

Thank you for publishing the programs I submitted and for your kind words to us down here in Southern Africa. I enjoy your reviews of the various SBC. I also purchased the Sardis SBC and I must say that it is a good company to deal with as they have answered all my letters promptly and are helping me in getting difficult-to-get parts here in "Darkest Africa". My only wish is that they implement a 2 MHz modification soon. Anyway once I have it up and running I may be motivated enough to let you have my comments on it. Do you know if the Artisan SBC (68 Micro, July '83 p. 56) is still alive and well?

Keep up the good work, as I really look forward to "OUR" magazine every month; pity it isn't a weekly!

Peter Piacenza

Dr. L. Piacenza.

Dear Sirs,

I herewith send you a copy of an advertisement from 1983, which seems to announce the outstanding DP-09-board for OS-9 operation, in which I am extreme interesseted to have. Last year I wrote a letter therefore to Artisan System Corp. for getting more information, but I didn't have any answer back till the present time. Do you know possibly more about this board in conjuction with OS-9 Level One? Is this corporation still alive and how can I get under which conditions this good-looking board? Yes, I do really hope that you might have more infos with your nice 68 Micro Journal about such a powerfull board, which I intent to run with two pieces of the wellknown 8" Siemens FD-100-8 floppydrives. Can any reader of your magazine give me helpful direction how I can come to the ownership of any DP-09-board? Let me thank you very much by this way for your search.

Hope you will send me some succesful lines to solve my sleepcatching problems with the DP-09. Thanks for all.

Yours sincerely.

Karl du Roi
Goslarsche Str. 33 b
D-3300 Braunschweig
Fed. Rep. of Germany

Editor's Note: Lately I have attempted to contact Artisan for information. First, the original telephone number (listed in past advertising) is reported - "non-working", meaning it is probably disconnected. Secondly, I have another number they gave us but I have called numerous times - no answer.

Based on that and the fact that I can never recommend you purchase anything from a company who may not be around long enough to give you proper service and support.

As far as OS-9 Support is concerned, there are several listed below that do support OS-9, as well as FLEX and Star-DOS and have had no complaints (with us) concerning quality, service or support. Any one of the three would be a good investment, as we have built them all and use them daily.

PT-69 Peripheral Technology

ST-2900 Sardis Technologies

UniBoard Digital Research Computers

Please note present address of each above in 68 Micro Journal advertising. Also reference is made to Single Board Computers SBC - a series of articles on each, a few months past.

DMW

- - -

Central Oklahoma Computer Organization, Inc.
1726 W. Roseoak Drive
Mustang, OK 73064

SIR:

Please publish the following information in your next issue. Hopefully your readers will make use of our bulletin board system and meetings. If you maintain a file of computer bulletin board systems, please delete any reference to "FLEXNET", a BBS formerly located in the Oklahoma City area. That system is no longer in operation, as the equipment was purchased by COCO INC. and is now used as the "COCONET".

- - - : = N O T I C E = : - - -

The Central Oklahoma Computer Organization, Inc. is a 278 member Radio Shack Color Computer users group. It meets the second Saturday of each month at 9:00 AM at 10th Street and Hudson in Oklahoma City. The club operates an open forum bulletin board system, the "COCONET" which may be reached at Ph. 1-405-376-1494, 24 hours a day, 7 days a week. The system contains COCO and FLEX operating system programs for download with no user connect fees.

- - - : = % % = : - - -

Robert Helms

Robert Helms, Vice-Chairman
434 West Ercoupe Drive
Midwest City, OK 73110
Tel. 1-405-733-3429

Dear Mr. Williams,

Please find enclosed, as you requested, the diskette containing the source for the routines to write and read the AAA Chicago Computer Companys Clock Calender Board, as well as the article itself.

I would like to thank you for extending my subscription, I would like to say though that it was unnecessary. I am more than happy to pay the subscription costs in order to receive your magazine. I am in a position to read many other journals and magazines that pertain to various computers systems, and truly feel that yours has more to offer a subscriber than a lot of the others combined.

Thanks again, have a prosperous year.

Regards

Clay Cederstrand

Clay Cederstrand

UNIVERSITY OF MAINE *at Farmington*

DEPARTMENT OF SCIENCES AND MATHEMATICS
Biology
Chemistry
Geology
Mathematics
Physics

39 High Street, Preble Hall
Farmington, Maine 04938
207-778-3501

Dear Don:

My article "Analog to Digital Digital to Analog" which you published in the March issue was incomplete. At first I panicked thinking that I had left out some of it. However, upon examining the disk AD_ARTCL.TXT, I found that the omission was not mine.

I have enclosed the disk and the hard copy printed from it. The portion outlined in red on pages 5 and 6 of the print-out was the part inadvertently omitted. The outlined portion is slightly more than one print-out page in length and begins with 180 DA% = PEEK ... and ends with INTEGER AD_OUT:

I hope you will publish the portions of these two programs which were omitted, but at the very least, I would appreciate a note saying the portions of the programs, obviously omitted, were not the fault of the author.

Thanks for the Journal, I look forward to receiving it each month.

J. Albert McDaniel

```
170 POKE HEX ("E11E"),CV%       : REM SET UP CHANNEL NUMBER
180 DA% = PEEK(HEX("E11C"))     : REM CLEARS CONVERSION
185 REM                           COMPLETE FLAG
190 POKE HEX ("E11F"),HEX("34") : REM TURN ON CONVERT PULSE
200 POKE HEX ("E11F"),HEX("3C") : REM TURN CONVERT PULSE OFF
210 ST% = PEEK(HEX("E11D"))     : REM READ STATUS
220 IF ST% < 128 THEN 210       : REM WAIT UNTIL HI BIT IS ON
230 DA%=(PEEK(HEX("E11E")) AND 15) * 256 + PEEK(HEX("E11C"))
240 IF DA% > 2047 THEN DA% = DA% - 4096
245 REM ADJUST FOR NEGATIVE VALUES
250 RETURN
260  REM NOW YOU MAY ADD THE VALUE OBTAINED TO A
270  REM REAL VARIABLE FOR SUMMING AND A AVAGING
280  REM PROGRAMS.
290  REM
300  REM TEST PROGRAM
310 GOSUB 100
340 CV% = 0 : REM SET CHANNEL TO ZERO
350 GOSUB 160
360 PRINT DA%
365 GOTO 340
370 END


10      REM DRIVER FOR 12 BIT A/D  CONVERTER IN BASIC
15      REM 0-5 V INPUT 0=0000,  5=4095

20      REM PORT ADDRESS IS 6E11X0 ON SAMPLE PROGRAM, BUT
30      REM MAY BE CHANGED TO ANY CONVENIENT I/O ADDRESS
40      REM CV% IS Variable FOR CHANNEL NUMBER OF A/D
100     REM TEST PROGRAM ** BY J. A. MCDANIEL **
110 GOSUB 210: REM INITIALIZE PORT
120 INPUT " THE CHANNEL NUMBER IS ", CV%
125 CV% = CV% * 16: REM CALCULATE CHANNEL NUMBER
130 GOSUB 320: REM PERFORM CONVERSION
140 PRINT DA%
150 GOTO 130
160 END
170     REM
200     REM FIRST THE PORT INITIALIZE SUBROUTINE
210 POKE HEX ("E11D"),0 : POKE HEX ("E11F"),0
220 POKE HEX ("E11C"),0 : POKE HEX ("E11E"),HEX ("F0")
230 POKE HEX ("E11D"),4 : POKE HEX ("E11F"),HEX ("3C")
240 RETURN              : REM THIS IS A SUBROUTINE
250       REM
300       REM  THE CONVERT SUBROUTINE
320 POKE HEX ("E11E"),CV%       : REM SET UP CHANNEL NUMBER
330 DA% = PEEK (HEX("E11C"))    : REM CLEARS CONVERSION COMPLETE
FLAG
340 POKE HEX ("E11F"),HEX("34") : REM TURN ON CONVERT PULSE
350 POKE HEX ("E11F"),HEX("3C") : REM TURN CONVERT PULSE OFF
360 ST% = PEEK (HEX("E11D"))    : REM READ STATUS
370 IF ST% < 128 THEN 210       : REM WAIT UNTIL HI BIT IS ON
380 DA%=PEEK(HEX("E11E")) AND 15: REM GET 4 MSBS REMOVE CH
NUMBER
390 BD%=NOT (DA%) AND 8 : REM COMPLEMENT UNCOMPLEMENTED BIT(
MSB)
400 BC%=  A% AND 7: REM GET COMPLEMENTED BITS ( 3 LSBS OF MSB)
410 DA%= BD% OR BC%: REM PUT BITS TOGETHER
420 BD%=PEEK(HEX("E11C"))
430 DA%=DA%*256 + BD%: REM TRANSFORM 2 8 BIT BIN. TO 12 BIT
440 RETURN: REM   CONVERT SUBROUTINE FINISHED

/* AD_CALBT TEST ROUTINE AD READS VOLTAGE ON A SCALE OF 0 TO 5
*/
/* PRINTS OUT THE HEX VALUE AND NOT(HEX VALUE) FROM THE A-D */
/* THEN PRINTS OUT THE VOLTAGE FROM 0.0000 TO 5.0000 */
/* ALSO AVERAGES 21 READINGS AND PRINTS THE RESULT */
/* BY J. A. MCDANIEL, UNIV. MAINE AT FARMINGTON */

ORIGIN =0000;
STACK = 0BDFF;

GLOBAL      BYTE DATA_A, DATA_B;
            BYTE ERFLAG,KEYCHAR;
            INTEGER AD_OUT;
            BYTE MS_BITS;
```

Ed's Note:

Albert, YOUR RIGHT! The omission was **not your fault.** The blame falls squarely on me. My apologies to you and the readers. Thank you for keeping me straight.

LEW

Larry E. Williams
'68 Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

OSU The Ohio State University

Department of Physiology
4196 Graves Hall
333 West 10th Avenue
Columbus, Ohio 43210-1239
Phone 614-422-5448

Dear Lew,

I am working on a paper for your journal and have run into some difficulty with the new format standards. I am using an original A mother board 6800 and have the old $14.95 assembler-editor from SWTPc, with tape storage. To output the assembler listing at 4½" would mean that I can't add any comments to my programs since there is only a fixed output format with this assembler and I can only make shorter lines by leaving out the comments. I appreciate the effort to make listings more readable because I have had some trouble reading some listings in the old format. As you can see I have no problem with text files since I have available an Apple computer with a word processor that formats easily. Do you have any suggestions for authors about how to handle listings (i.e., leave off comments, make short comments, buy a new assembler, etc.).

Your magazine is a unique source of information for us 68XX types - keep up the good work.

Thanks,

Keith Michal

Lew's Reply,

Keith,

I don't have any hard and fast solutions to keeping the longest line in a source listing to a maximun of 4 1/2". I don't feel that authors should leave out any portion of the listing just to satisfy my line width requirement. I would insist that nothing which normally is included in a source listing be excluded just to hold a line length. Holding text column widths to 4 1/2" should be no problem for 99% of the people submitting material. I am requesting only that if it is possible to do the same with the source listing, then please do so. If there are several lines in the source listing that can not get down to 4 1/2", then that's O.K.. I may have to work with the listing a little more carefully, but I would much rather have to do a little more work than not have it at all. Just because not every source listing can be held to 4 1/2", will in NO way jeopardize the publication of that article. The new line width request is simply an attempt to leave all characters as large as possible in a single page column for easier reproduction and reading.

I thank you again for doing all you have done to help me out. I appreciate your efforts.

Lew E. Wall

LEW

Classified Advertising

TELETYPE Model 43 PRINTER - with serial (RS232) interface, and full ASCII keyboard. **LIKE NEW** - New cost $1295.00 - ONLY **$759.00** ready to run - Call Tom - Larry - Bob, CPI 615 842-4600

***

SWTPc 6809 w/MF-68, 56K, DMAF1, AC-30, ACW VB & keyboard, NewTech Music Board, 2 MP-S, MP-LA, MP-R, H-14 printer, Adventure, Scredltor III, Pascal, Forth, XBasic, Sort/Merge, 6809 Debug, Diagnostics, Asmb, PR, FLEX Programmers Manual, $1200 obo
Robert H. Morrison, Box 912, APO NY 09057-5363.

***

Wanted to Buy: Used 6800 and 6809 FLEX software. Commercial and Public Domain. Send description and asking price to:
John Current P.O.Box 273 Honolulu, HI 96859

***

SSB 2MHz 6809 4Meg Hard, DSDD Floppy, 96K, OS9, INTKOL-C, BASICO9, much more. Best offer. Also DSDD drive, 16K, SS-50 Chassis, Swptc boards, Terminals, MX-80.
Robert (415)775-6982

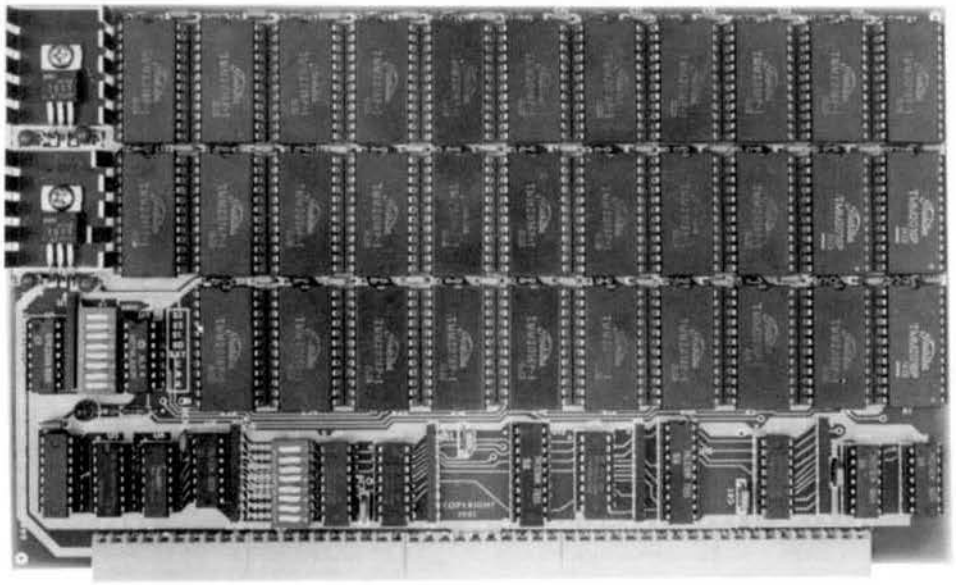
64K SS-50 STATIC RAM

PRICE CUT!!

$119.00 (48K KIT)

NEW!

LOW POWER!



RAM OR EPROM!

BLANK PC BOARD WITH DOCUMENTATION $45

SUPPORT ICs + CAPS -	$18.00
FULL SOCKET SET -	$15.00

ASSEMBLED AND TESTED **ADD $50**

FEATURES:

- ★ Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
- ★ Fully supports Extended Addressing.
- ★ 64K draws only approximately 500 MA.
- ★ 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
- ★ Board is configured as 3-16K blocks and 8-2K blocks (within any 64K block) for maximum flexibility.
- ★ 2716 EPROMs may be installed anywhere on Board.
- ★ Top 16K may be disabled in 2K blocks to avoid any I/O conflicts.
- ★ One Board supports both RAM and EPROM.
- ★ RAM supports 2MHZ operation at no extra charge!
- ★ Board may be partially populated in 16K increments.

56K	$129
64K	$139

CLOSE OUT SPECIAL

WE HAVE DROPPED OUR 32K SS-50 STATIC RAM BOARD WHICH USED 2114 LOW POWER RAMS. WE WILL SELL THE REMAINING STOCK OF BLANK PCB'S WITH DATA FOR $17.50 EA. THESE FORMERLY SOLD FOR $50.

16K STATIC RAMS?

The new 2K x 8, 24 PIN, static RAMs are the next generation of high density, high speed, low power, RAMs. Pioneered by such companies as HITACHI and TOSHIBA, and soon to be second sourced by most major U.S. manufacturers, these ultra low power parts, feature 2716 compatible pin out. Thus fully interchangeable ROM/RAM boards are at last a reality, and you get BLINDING speed and LOW power thrown in for virtually nothing.

Digital Research Computers
(OF TEXAS)
P.O. BOX 461565 • GARLAND, TEXAS 75046 • (214) 225-2309

TERMS: Add $2.00 postage. We pay balance. Order under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50, add 85¢ for insurance.

WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: 2758, 2508, 2716, 2516, 2732/2732A, MCM68766/6, 2766/2766A, 2256, 27128/27128A, and 27256. (I=Intel, T=Texas, M=Motorola.

NO PERSONALITY MODULES REQUIRED!

TRI-VOLT EPROMS ARE NOT SUPPORTED

INTEL's inteligent programming (tm) implemented for Intel 2764, 27128 and 27256 devices. Intelligent programming reduces the average programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

Fully enclosed pod with 5' of flat ribbon cable for connection to the host computer MC6821 PIA interface board.

MC6809 software for FLEX and OS9 (Level 1 or 2, Version 1.2).

BINARY DISK FILE offset loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

a. FILL ..... a selected area of the buffer with a HEX char.
b. MOVE ..... blocks of data.
c. DUMP ..... the buffer in HEX and ASCII.
d. FIND ..... a string of bytes in the buffer.
e. EXAMINE/CHANGE . the contents of the buffer.
f. CRC ..... checksum a selected area of the buffer.
g. COPY ..... a selected area of an EPROM into the buffer.
h. VERIFY ..... a selected area of an EPROM against the buffer.
i. PROGRAM ..... a selected area of an EPROM with data in the buffer.
j. SELECT ..... a new EPROM type (return to types menu).
k. ENTER ..... the system monitor.
l. RETURN ..... to the operating system.
m. EXECUTE ..... any DOS utility (only in FLEX and OS9 versions).

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX. SSB/MDOS CONTACT US DIRECT.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-deDugger, which, amongst other things, can single step the program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or license fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALs.
- Vectors (single dimension arrays) and pointers are supported.
- Mathematical expressions: (+), (-), (*), (/), modulus (\), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical operators: (.AND), (.OR), (.EOR/XOR)
- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (XREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute cinch!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BITIO, HARDIO, HEXIO, FLEXIO, SCIPACK, STRSUBS, BASTRING, and REALCON.

'... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE.'

Quoted from Ron Andersons FLEX User Notes column in '68. Need we say more?

MACE/XMACE/ASM05

ALL of these products feature a highly productive environment where the editor and the assembler reside in memory together. Gone are the days of tedius disk load and save operations while you are debugging your code.

- Friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also produce GASPROCS (GEN statements) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 6800/1/2/3/8 and supports the extended mnemonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step tracer and mini in-line disassembler that is easy to use?? Look no further, you have found it. This package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 6K (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD it, Aim it and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX VII and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficient assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler. The TSC relocating assembler is only required if you want to generate your own libraries.
- The pre-processor, compiler, optimizer, assembler and loader all run independantly or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:

- Talker	- Serial Poll	- Single or Dual Primary Address
- Listener	- Parallel Poll	- Secondary Address
- System Controller	- Group Trigger	- Talk only ... Listen only

- Fully documented with a complete reprint of the BYTE article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6809 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-30 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(6809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-II/U	with one version of software (no cable or interface)	$395.00
UPROM-II/C	as above but complete with cable and S-30 interface	$545.00
CABLE	5' twist-n-flat 50 way cable with IDC connectors	$ 35.00
S-30 INT	S5-30 interface for UPROM-II	$130.00
EXOR INT	Motorola EXORbus (EXORciser) interface for UPROM-II	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEX or OS9 AND disk size?	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

ALL PRICES INCLUDE AIR MAIL POSTAGE

Terms: CWO. Payment by Int'l Money Order, VISA or MASTER-CARD also accepted.

WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK, ENGLAND. NR28 9SA.

TEL: 44 (692) 404086
TLX: 975548 WMICRO G

WE STOCK THE FOLLOWING COMPANIES PRODUCTS: GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O, & ALFORD & ASSOCIATES.

FLEX (tm) is a trademark of Technical Systems Consultants, OS-9 (tm) is a trademark of Microware Systems Corporation, MDOS (tm) and EXORciser (tm) are trademarks of Motorola Incorporated.

68' MICRO JOURNAL ADVERTISERS INDEX

This index is provided as a reader service. The publisher does not assume any liability for omissions or errors.

GIMIX HAS THE 6809 SYSTEM TO SUIT YOUR NEEDS

HARDWARE

All systems feature the **GIMIX CLASSY CHASSIS**; with a ferro-resonant constant voltage power supply, gold plated bus connectors, and plenty of capacity for future expansion.

Static **RAM** and double-density **DMA** floppy disk controllers are used exclusively in all systems.

All systems are guaranteed for 2 MHz operation and include complete hardware and software documentation, necessary cables, filler plates, etc.

Systems are assembled using burned-in and tested boards, and all disk drives are tested and aligned by **GIMIX**.

You can add additional components to any system when ordering, or expand it in the future by adding **RAM, I/O,** etc.

GIMIX lets you choose from a wide variety of options to customize your system to your needs.

SOFTWARE

All **DS-9/FLEX** systems allow you to software select either operating system.

Also included is the **GMXBUG** monitor and, in systems with 128K or more of RAM, **GMX-VDISK** for **FLEX**.

All **GIMIX OS-9** systems include **Microware's Editor, Assembler, Debugger, Basic09,** and **Runb**; and the GMX versions of **RMS** and **DO** for **OS-9**.

All **GIMIX** versions of **OS-9** can read and write RS color computer format OS-9 disks, as well as the Microware/GIMIX standard format.

New and exclusive with **OS-9 GMX III** systems is the **GMX OS-9 Support ROM**, a monitor for **OS-9** that includes memory diagnostics and allows the system to boot directly from either hard disk or floppy.

A wide variety of languages and other software is available for use with either **OS-9** or **FLEX**.

OS-9 GMX III/FLEX SYSTEMS (#79)

The **#79** super system now includes (in addition to the above): the **GMX 6809 CPU III, a 256K CMOS Static RAM Board (#72), and a 3-port Intelligent Serial I/O Processor (#11)**.

The **GMX 6809 CPU III** can perform high-speed DMA transfers from memory to memory and uses memory attributes and illegal instruction trapping to protect the system and users from program crashes. If a user program crashes, only that user is affected; other users are unaware of the problem.

The 3-Port Intelligent Serial I/O Board (#11) significantly reduces system overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud.

with dual 40 track DSDD drives	**$5998.79**
with dual 80 track DSDD drives	**$6198.79**
with **#88** dual 8" DSDD drivesystem	**$7698.79**
with **#90** 19MB Winchester subsystem and one 80 track	**$8898.79**
with a 47MB Winchester subsystem and one 80 track	**$10,698.79**
with a 47MB plus a 6MB removable pack Winchester subsystem and one 80 track drive	**$12,398.79**

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account #73-32033.

OS-9 GMX I / FLEX SYSTEMS #49

The **#49** systems include 64KB static RAM, #05 CPU, #43 2 port serial board.

with dual 40 track OSDD drives	**$3998.49**
with dual 80 track DSDD drives	**$4198.49**
with **#88** dual 8" DSDD drive system	**$5698.49**
with **#90** 19MB Winchester subsystem and one 80 track	**$6898.49**

OS-9 GMX II / FLEX SYSTEMS #39

The **#39** systems include 128KB static RAM, #05 CPU, #43 2 port serial board.

with dual 40 track DSDD drives	**$4498.39**
with dual 80 track DSDD drives	**$4698.39**
with **#88** dual 8" DSDD drive system	**$6198.39**
with **#90** 19MB Winchester subsystem and one 80 track	**$7398.39**

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

ALL PRICES ARE F.O.B. CHICAGO

Contact GIMIX for price and availability of UniFLEX and UniFLEX GMXIII Systems.

NOTE on all drive systems: Dual 40 track drives have about 700KB of formatted capacity; dual 80's about 1,400KB; dual 8" about 2,000KB. The formatted capacity of hard disks is about 80% of the total capacity.

Want to expand your system to a megabyte of Static RAM and 15 users?

Simply add additional memory and I/O boards. Your GIMIX system can grow with your needs. Contact us for a complete list of available boards and options.

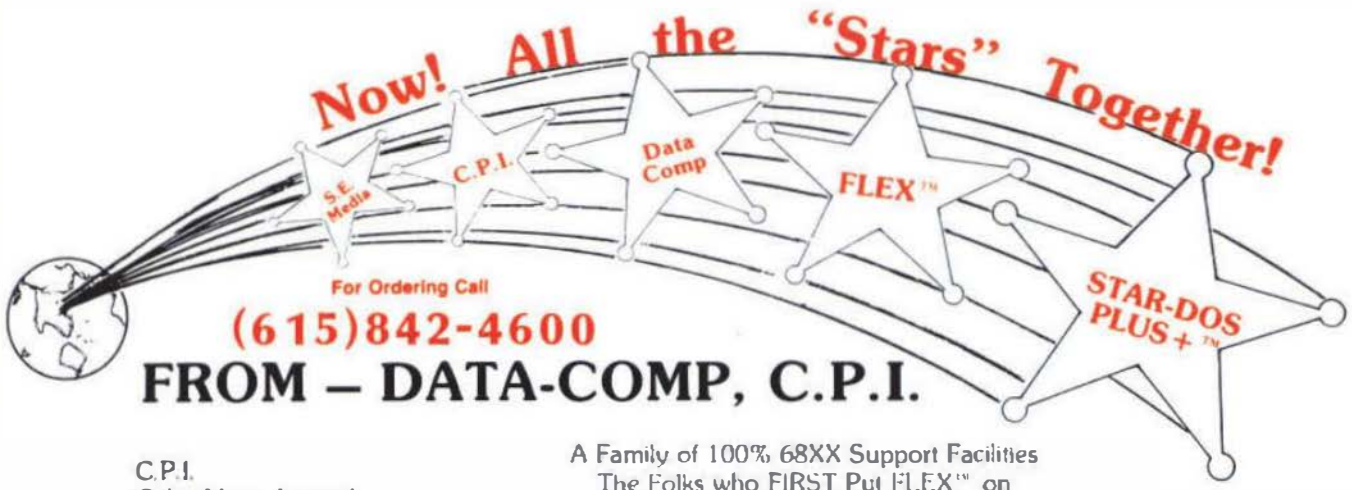
#72 256KB CMOS STATIC RAM board with battery back up	**$1098.72**
#64 64KB CMOS STATIC RAM board with battery back up	**$528.64**
#67 64KB STATIC RAM board	**$478.67**
#11 3 port intelligent serial I/O board	**$498.11**
#43 2 port serial I/O board	**$128.43**
#42 2 port parallel I/O board	**$88.42**
#95 cable sets (1 needed per port), specify board	**$24.95**

NOW SHIPPING !

UniFLEX

GMX III Systems

GIMIX INC.
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055



For Ordering Call

(615)842-4600

FROM – DATA-COMP, C.P.I.

C.P.I.
Color Micro Journal
'68' Micro Journal
Data-Comp
S.E. Media

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo
Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+ ™

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00
Only $79.95

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.50

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ Super 800 Support
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

NEW LOWER PRICES ON PAK #5, AND PRINTERS

THESE PACKAGES INCLUDE DRIVE, *CONTROLLER, POWER SUPPLY & CABINET, CABLE, AND MANUAL.

* SPECIFY WHAT CONTROLLER YOU WANT J&M, OR RADIO SHACK.

PAK #1	1 SINGLE SIDED, DOUBLE DENSITY SYS.	$349.95
PAK #2	2 SINGLE SIDED, DOUBLE DENSITY SYS.	$639.95
PAK #3	1 DOUBLE SIDED, DOUBLE DENSITY SYS.	$439.95
PAK #4	2 DOUBLE SIDED, DOUBLE DENSITY SYS.	$699.95
PAK #5	2 DOUBLE SIDED, DOUBLE DENSITY SYS. THINLINE DRIVES, HALF SIZE	$499.99

Controllers

J&M DISK CONTROLLER W/ JDOS OR RADIO SHACK DISK BASIC, SPECIFY WHAT DISK BASIC.	$134.95
RADIO SHACK DISK CONTROLLER 1.1	$134.95

Misc.

64K UPGRADE W/MOD. INSTRUCTIONS, C,D,E,F, AND COCO 2	$ 44.95
HJL KEYBOARDS	$ 74.95
MICRO TECH LOWER CASE ROM ADAPTER	$ 74.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET & POWER SUPPLY	$ 49.95
SINGLE SIDED, DOUBLE DENSITY 5" DISK DRIVE	$199.95
DOUBLE SIDED, DOUBLE DENSITY 5" DISK DRIVE	$249.95

Printers

EPSON RX-80	$269.00
EPSON RX-80FT	$369.00
EPSON MX-100	$499.00
EPSON FX-100	$799.00
EPSON FX-80	$549.00
EPSON MX-70	$200.00

Disk Drive Cables

CABLE FOR ONE DRIVE	$ 19.95
CABLE FOR TWO DRIVES	$ 24.95

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
TELEX 558 414 PVT BTH

68 Micro Journal
ISSN 0194-5025
5900 Cassandra Smith Rd.
Hixson, TN 37343

Second Class Postage Paid
At Hixson, TN
And Additional Mailing Offices